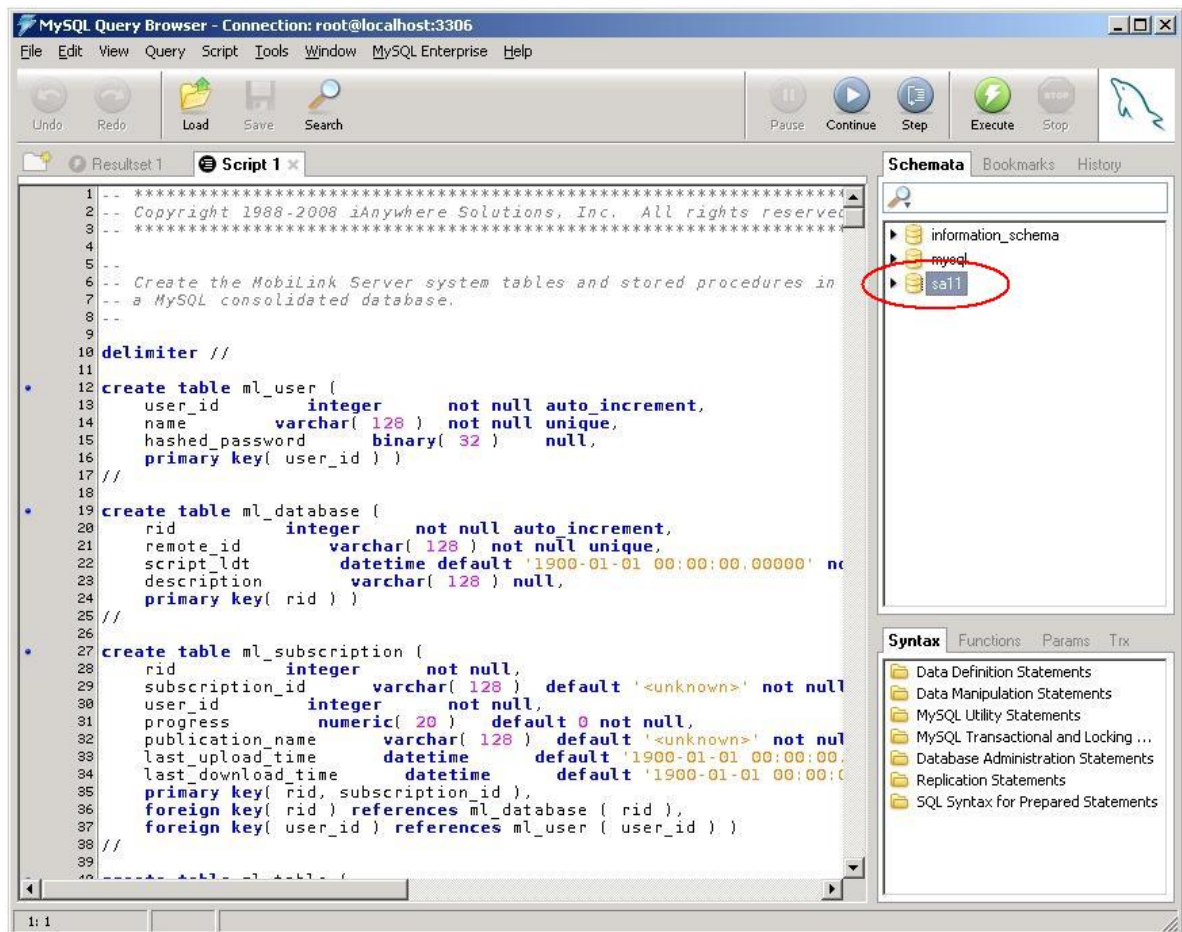


## Introducción a MySQL Stores Procedures



Para poder realizar los ejemplos de este documento se necesita:

1. MySQL 5.0 o superior
2. Conocimientos de SQL básicos
3. Conocimientos de programación en algún language
4. Un cliente MySQL (para este tutorial se ha utilizado SQLYog)

Intro: Procedimientos almacenados en MySQL

MySQL soporta el uso de “Rutinas”, hay dos tipos de rutinas: Procedimientos Almacenados, que ejecutan una serie de sentencias y pueden devolver valores. Funciones, que devuelven un valor (como pi(), max(), min(), etc).

## ¿Por qué Procedimientos Almacenados?

1. Son tecnología probada 2. Son rápidos 3. Son componentes 4. Son portables 5. Están almacenados 6. Son Migrables

## Crear una base de datos de pruebas con datos:

```
CREATE DATABASE db5;  
USE db5;  
CREATE TABLE t(s1 INT);  
INSERT INTO t VALUES (5);
```

## Establecer un delimitador

```
DELIMITER //;
```

En MySQL las sentencias se ejecutan luego de escribir el signo punto y coma (;), por esta razón antes de escribir el procedimiento almacenado la función del punto y coma se asigna a otros caracteres usando la sentencia *DELIMITER* seguida de un carácter tal como |, de esta manera el procedimiento puede ser escrito usando los punto y comas sin que se ejecute mientras se escribe; después de escrito el procedimiento, se escribe nuevamente la sentencia *DELIMITER* ; para asignar al punto y coma su función habitual.

## Ejemplo de create procedure:

```
CREATE PROCEDURE p1 () SELECT * FROM t; //
```

Los nombres de procedimientos no son case-sensitive, p1() y P1() son el mismo procedimiento.

Sentencias validas dentro de un procedimiento:

Todas las sentencias SQL tales como SELECT, DROP, INSERT , ETC. Sólo hay que recordar que si se utilizan funciones exclusivas o extensiones de MySQL el código no será 100% portable.

Sentencias NO válidas dentro de un procedimiento:

Todas las relativas al manejo y creación de procedimientos, CREATE PROCEDURE, ALTER PROCEDURE, CREATE TRIGGER, etc... Además, la sentencia USE database está prohibida. Llamada al procedimiento:

```
CALL p1();
```

s1

5

Crear un procedimiento con características adicionales:

```
CREATE PROCEDURE p2() #SENTENCIA DE CREACION  
LANGUAGE SQL #INFORMA DEL TIPO DE LANGUAGE EN CASO DE PORTABILIDAD  
NOT DETERMINISTIC #NO DEVUELVE SIEMPRE LOS MISMOS RESULTADOS YA QUE ES  
UNA SELECT  
SQL SECURITY DEFINER #INDICA AL SERVIDOR QUE DEBE COMPROBAR LOS  
PERMISOS DE USUARIO ANTES DE CREAR EL PROCEDIMIENTO  
COMMENT 'procedimiento de pruebas' #LINEA DE COMENTARIO ADICIONAL  
SELECT CURRENT_DATE, RAND() FROM t #SENTENCIAS SQL
```

```
CALL p2();
```

```
current_date rand()
```

---

```
2010-12-05 0.325902371970846
```

El servidor MySQL guarda las variables de entorno con cada procedimiento que se crea, de manera que aunque estas cambien, al llamar nuevamente al procedimiento dará el mismo resultado. Ejemplo:

```
SET sql_mode = 'ansi'; #CAMBIO EL MODO SQL  
CREATE PROCEDURE p3() SELECT 'a' || 'b'; #CREO EL PROCEDIMIENTO  
SET sql_mode=""; #CAMBIO EL MODO SQL Y LO DEJO COMO ESTABA  
CALL p3(); #LLAMO A p3()
```

```
'a' || 'b'
```

---

```
ab
```

Aunque el mode SQL se haya cambiado inmediatamente después de creado el procedimiento, la llamada devuelve los valores en modo ansi, que es el modo en el momento de la creación del procedimiento.

## Parámetros:

*CREATE PROCEDURE p5() ... # Sin parámetros*

*CREATE PROCEDURE p5([IN] name data-type) #Un parámetro de entrada, la palabra reserva IN es opcional porque por defecto todos los parámetros son de entrada*

*CREATE PROCEDURE p5(OUT name data-type) #Un parámetro de salida*

*CREATE PROCEDURE p5(INOUT name data-type) #Un parámetro de entrada y salida*

## Ejemplo de parámetro IN

Este procedimiento crea una variable de session X que toma el valor que sea pasado como parámetro al procedimiento:

```
CREATE PROCEDURE p5(p INT)
```

```
SET @X = p
```

```
CALL p5(12345);
```

```
SELECT @X;
```

```
@x
```

---

```
12345
```

## Ejemplo de parámetro OUT

En este caso, la palabra OUT indica que el valor sale fuera del procedimiento, que es un valor retornado. Este valor será asignado a la variable @Y. El efecto es el mismo que llamar a SET @Y = -5;

*#Creo el procedimiento*

```
CREATE PROCEDURE p6 (OUT p INT)
```

```
SET p = -5;
```

#Lo llamo y asigno el valor a @y

CALL p6(@Y);

#Selecciono @y

SELECT @y;

@y

—

-5

## Extendiendo las sentencias dentro del procedimiento

Cada vez que exista más de una sentencia SQL dentro del procedimiento, se debe utilizar la estructura BEGIN .. END para crear bloques de sentencias, que funciona igual que las transacciones o los bloques en otros lenguajes como C o Java:

```
CREATE PROCEDURE p7 ()  
BEGIN  
SET @a = 5;  
SET @b = 5;  
INSERT INTO t VALUES (@a);  
SELECT s1 * @a FROM t WHERE s1>=@b;  
END;
```

## Sentencias SQL

### Variables

Para declarar variables se utiliza DECLARE. Las variables no se declaran dentro del procedimiento sino dentro de un bloque BEGIN/END. Estas no son variables de session, no comienzan por @. Se deben declara explícitamente junto con el tipo de datos.

```
DELIMITER //  
CREATE PROCEDURE p8 ()  
BEGIN  
DECLARE a INT;
```

```
DECLARE b INT;  
SET a = 5;  
SET b = 5;  
INSERT INTO t VALUES (a);  
SELECT s1 FROM t WHERE s1 >= b;  
END; //
```

El valor por defecto de las variables es NULL.

## Ejemplo con variables con valor por defecto:

```
DELIMITER //  
CREATE PROCEDURE p10 ()  
BEGIN  
DECLARE a, b INT DEFAULT 5;  
INSERT INTO t VALUES (a);  
SELECT s1 FROM t WHERE s1 >= b;  
END; //
```

CALL p10();

s1 \* a

---

25

25

## Condiciones IF THEN ELSE

```
DELIMITER $$
USE `db5` $$
DROP PROCEDURE IF EXISTS `p12` $$
CREATE PROCEDURE `p12`(IN parameter1 INT)
BEGIN
    DECLARE variable1 INT;
    SET variable1 = parameter1 + 1;
    IF variable1 = 0 THEN
        INSERT INTO t VALUES (17);
    END IF;
    IF parameter1 = 0 THEN
        UPDATE t SET s1 = s1 + 1;
    ELSE
        UPDATE t SET s1 = s1 + 2;
    END IF;
END;
```

CALL p12(0);

(5 row(s) affected)

s1

——

6

6

## Estructura CASE

Es una alternativa a IF para evaluar los posibles valores de una variable:

```
DELIMITER //
CREATE PROCEDURE p13 (IN parameter1 INT)
BEGIN
    DECLARE variable1 INT;
    SET variable1 = parameter1 + 1;
```

```
CASE variable1
WHEN 0 THEN INSERT INTO t VALUES (17);
WHEN 1 THEN INSERT INTO t VALUES (18);
ELSE INSERT INTO t VALUES (19);
END CASE;
END;
CALL p13(1);
```

(1 row(s) affected)

s1

\_\_\_\_\_

6

6

19

Si este mismo procedimiento se ejecuta pasando como valor un NULL, entonces entra en el ELSE final, insertando un 19:

```
CALL p13(NULL);
```

s1

\_\_\_\_\_

6

6

19

19



# Loops

Existen 3 estructuras iterativas standard: WHILE ... END WHILE LOOP ... END LOOP  
REPEAT ... END REPEAT y una no standard: GO

## WHILE ... END WHILE

```
CREATE PROCEDURE p14 ()  
BEGIN  
  DECLARE v INT;  
  SET v = 0;  
  WHILE v < 5 DO  
    INSERT INTO t VALUES (v);  
    SET v = v + 1;  
  END WHILE;  
END;
```

Si v no se inicializa a 0, entonces vale NULL. En ese caso, el bucle haría en cada vuelta NULL + 1 = NULL y nunca se saldría del bucle.

CALL p14();

s1

---

6

6

19

19

0

1

2

3

4

## REPEAT ... END REPEAT

Tiene el mismo funcionamiento que WHILE, excepto que comprueba la condición DESPUES y no antes como lo haría WHILE:

```
CREATE PROCEDURE p15()  
BEGIN  
  DECLARE v INT;  
  SET v = 0;  
  REPEAT  
    INSERT INTO t VALUES (v);  
    SET v = v + 1;  
  UNTIL v >= 5 #ojo, sin ; al final de la sentencia  
  END REPEAT;  
END;
```

```
CALL p15();
```

(1 row(s) affected)

```
SELECT COUNT(*) FROM t;
```

```
count(*)
```

---

## LOOP ... END LOOP (LOOP ... END LOOP con IF y LEAVE)

El bucle LOOP no requiere condiciones al principio ni al final, la condición se puede establecer en cualquier punto:

```
DELIMITER //
CREATE PROCEDURE p16()
BEGIN
DECLARE v INT;
SET v = 0;
loop_label: LOOP
INSERT INTO t VALUES (t);
SET v = v + 1;
IF v >= 5 THEN
LEAVE loop_label;
END IF;
END LOOP;
END;
CALL p16();
```

(1 row(s) affected)

```
SELECT COUNT(*) FROM t;
```

```
COUNT(*)
```

---

## Labels

```
CREATE PROCEDURE p17()  
label_1: BEGIN  
label_2: WHILE 0 = 1 DO LEAVE label_2; END WHILE;  
label_3: REPEAT LEAVE label_3; UNTIL 0 = 0  
END REPEAT;  
label_4: LOOP LEAVE label_4;  
END LOOP;  
END;
```

Es posible utilizar una etiqueta antes de BEGIN, WHILE, REPEAT o LOOP. La sentencia LEAVE label\_3 significa “salir del bloque o bucle identificado como label\_3”

## Etiquetas END

Se pueden utilizar etiquetas END label para evitar confusión y mejorar la legibilidad de los procedimientos.

```
CREATE PROCEDURE p18()  
label_1: BEGIN  
label_2: WHILE 0 = 1 DO LEAVE label_2; END WHILE label_2;  
label_3: REPEAT LEAVE label_3; UNTIL 0 = 0  
END REPEAT label_3;  
label_4: LOOP LEAVE label_4;  
END LOOP label_4;  
END;
```

## Leave y LABELS

Estas etiquetas tiene la utilidad, también, de sacarnos de sentencias IF anidadas en profundidad:

```
CREATE PROCEDURE p19(parameter1 CHAR)
label_1: BEGIN
label_2: BEGIN
label_3: BEGIN
IF parameter1 IS NOT NULL THEN
IF parameter1 = 'a' THEN
LEAVE label_1;
ELSE BEGIN
IF parameter1 = 'b' THEN
LEAVE label_2;
ELSE LEAVE label_3;
END IF;
END;
END IF;
END IF;
END;
END;
END;
```

## ITERATE

En este caso, la sentencia LABEL es necesaria. LABEL actúa como objetivo de ITERATE. En el siguiente ejemplo, ITERATE loop\_label, significa “comenzar de nuevo la etiqueta loop\_label”. Es equivalente a una sentencia continue en C o Java.

```

CREATE PROCEDURE p20()
BEGIN
DECLARE v INT;
SET v = 0;
loop_label: LOOP
IF v = 3 THEN
SET v = v + 1;
ITERATE loop_label;
END IF;
INSERT INTO t VALUES (v);
IF v >= 5 THEN
LEAVE loop_label;
END IF;
END LOOP;
END;

```

## GOTO

```

CREATE PROCEDURE p...
BEGIN
...
LABEL label_name;
...
GOTO label_name;
...
END;

```

MySQL soporta la sentencia GOTO aunque no es standard SQL y por eso no aparece en el manual. Esta sentencia se ha agregado por compatibilidad con otros DBMS.

## FUNCIONES EN MYSQL

MySQL tiene muchas funciones que podemos usar en nuestro procedimientos almacenados y consultas, pero en ocasiones podemos necesitar crear **nuestras propias funciones** para hacer cosas más especializadas.

Vamos a ver **cómo crear funciones** en MySQL:

```
DELIMITER // CREATE FUNCTION holaMundo() RETURNS VARCHAR(20) BEGIN  
RETURN 'HolaMundo'; END //
```

Para comprobar que funciona **tecleamos lo siguiente** en la consola de MySQL :

```
Select holaMundo();
```

Lo que devuelve el siguiente resultado :

```
mysql> select holaMundo()// +-----+ | holaMundo() | +-----+ | Hola  
Mundo!! | +-----+ 1 row in set (0.00 sec)
```

Para **borrar la función** que acabamos de crear :

```
DROP FUNCTION IF EXISTS holaMundo
```

## Uso de las variables en funciones:

Las variables en las funciones se usan de igual manera que en los [procedimientos almacenados](#), se declaran con la sentencia **DECLARE**, y se asignan valores con la sentencia **SET**.

```
DELIMITER // CREATE FUNCTION holaMundo() RETURNS VARCHAR(30) BEGIN  
DECLARE salida VARCHAR(30) DEFAULT 'Hola mundo'; ; SET salida = 'Hola mundo con  
variables'; RETURN salida; END //
```

Esta variable es de **ámbito local**, y será destruida una vez finalice la función. Cabe destacar el uso de la sentencia DEFAULT en conjunto con DECLARE, que asigna un valor por defecto al declarar la variable.

## Uso de parámetros en funciones:

```
DROP FUNCTION IF EXISTS holaMundo CREATE FUNCTION holaMundo(entrada
VARCHAR(20)) RETURNS VARCHAR(20) BEGIN DECLARE salida VARCHAR(20); SET
salida = entrada; RETURN salida; END
```

Ahora hemos creado una función que devuelve el mismo valor que le pasamos como parámetro.

Si tecleamos :

```
mysql> select holaMundo("nosolocodigo")// +-----+ |
holaMundo("nosolocodigo") | +-----+ | nosolocodigo | +-----+
-----+ 1 row in set (0.00 sec)
```

Obtenemos como resultado lo mismo que le hemos pasado como parámetro, en este caso “nosolocodigo”

Para finalizar, algo un poco más complejo, vamos a crear una **función que acepte un dividendo y un divisor y haga una división sin usar el operador división:**

```
create function divide(dividendo int, divisor int) returns int
begin
    declare aux int; declare contador int; declare resto int;
    set contador = 0; set aux = 0;
    while (aux + divisor) <= dividendo do
        set aux = aux + divisor ; set contador = contador + 1;
    end while;
    set resto = dividendo - aux ;
    return contador;
end; //
```

Para usarlo, simplemente **llamaríamos a la función así:**


```
SELECT divide(20,2)
```

Lo que **devolvería 10.**



# FUNCIONES PREDEFINIDAS EN MYSQL


## Funciones de Control de Flujo

 **Funciones de control de flujo**

Las funciones de esta categoría son:

<u>IF</u>	Elección en función de una expresión booleana
<u>IFNULL</u>	Elección en función de si el valor de una expresión es <i>NULL</i>
<u>NULLIF</u>	Devuelve <i>NULL</i> en función del valor de una expresión

## Funciones Matemáticas

 **Funciones matemáticas**

Las funciones de la categoría de matemáticas son:

<u>ABS</u>	Devuelve el valor absoluto
<u>ACOS</u>	Devuelve el arcocoseno
<u>ASIN</u>	Devuelve el arcoseno
<u>ATAN y ATAN2</u>	Devuelven el arcotangente
<u>CEILING y CEIL</u>	Redondeo hacia arriba
<u>COS</u>	Coseno de un ángulo
<u>COT</u>	Cotangente de un ángulo
<u>CRC32</u>	Cálculo de comprobación de redundancia cíclica
<u>DEGREES</u>	Conversión de grados a radianes
<u>EXP</u>	Cálculo de potencias de <i>e</i>
<u>FLOOR</u>	Redondeo hacia abajo
<u>LN</u>	Logaritmo natural
<u>LOG</u>	Logaritmo en base arbitraria
<u>LOG10</u>	Logaritmo en base 10
<u>LOG2</u>	Logaritmo en base dos
<u>MOD o %</u>	Resto de una división entera
<u>PI</u>	Valor del número $\pi$
<u>POW o POWER</u>	Valor de potencias
<u>RADIANS</u>	Conversión de radianes a grados
<u>RAND</u>	Valores aleatorios
<u>ROUND</u>	Cálculo de redondeos
<u>SIGN</u>	Devuelve el signo
<u>SIN</u>	Cálculo del seno de un ángulo
<u>SQRT</u>	Cálculo de la raíz cuadrada
<u>TAN</u>	Cálculo de la tangente de un ángulo
<u>TRUNCATE</u>	Elimina decimales

## Funciones de Cadenas de Caracteres

## Funciones de cadenas

Las funciones para tratamiento de cadenas de caracteres son:

<u>ASCII</u>	Valor de código ASCII de un carácter
<u>BIN</u>	Conversión a binario
<u>BIT_LENGTH</u>	Cálculo de longitud de cadena en bits
<u>CHAR</u>	Convierte de ASCII a carácter
<u>CHAR_LENGTH</u> o <u>CHARACTER_LENGTH</u>	Cálculo de longitud de cadena en caracteres
<u>COMPRESS</u>	Comprime una cadena de caracteres
<u>CONCAT</u>	Concatena dos cadenas de caracteres
<u>CONCAT_WS</u>	Concatena cadenas con separadores
<u>CONV</u>	Convierte números entre distintas bases
<u>ELT</u>	Elección entre varias cadenas
<u>EXPORT_SET</u>	Expresiones binarias como conjuntos
<u>FIELD</u>	Busca el índice en listas de cadenas
<u>FIND_IN_SET</u>	Búsqueda en listas de cadenas
<u>HEX</u>	Conversión de números a hexadecimal
<u>INSERT</u>	Inserta una cadena en otra
<u>INSTR</u>	Busca una cadena en otra
<u>LEFT</u>	Extraer parte izquierda de una cadena
<u>LENGTH</u> u <u>OCTET_LENGTH</u>	Calcula la longitud de una cadena en bytes
<u>LOAD_FILE</u>	Lee un fichero en una cadena
<u>LOCATE</u> o <u>POSITION</u>	Encontrar la posición de una cadena dentro de otra
<u>LOWER</u> o <u>LCASE</u>	Convierte una cadena a minúsculas
<u>LPAD</u>	Añade caracteres a la izquierda de una cadena
<u>LTRIM</u>	Elimina espacios a la izquierda de una cadena
<u>MAKE_SET</u>	Crea un conjunto a partir de una expresión binaria
<u>OCT</u>	Convierte un número a octal
<u>ORD</u>	Obtiene el código ASCII, incluso con caracteres multibyte
<u>QUOTE</u>	Entrecomilla una cadena
<u>REPEAT</u>	Construye una cadena como una repetición de otra
<u>REPLACE</u>	Busca una secuencia en una cadena y la sustituye por otra
<u>REVERSE</u>	Invierte el orden de los caracteres de una cadena
<u>RIGHT</u>	Devuelve la parte derecha de una cadena
<u>RPAD</u>	Inserta caracteres al final de una cadena
<u>RTRIM</u>	Elimina caracteres blancos a la derecha de una cadena
<u>SOUNDEX</u>	Devuelve la cadena "soundex" para una cadena concreta
<u>SOUNDS LIKE</u>	Compara cadenas según su pronunciación
<u>SPACE</u>	Devuelve cadenas consistentes en espacios
<u>SUBSTRING</u> o <u>MID</u>	Extraer subcadenas de una cadena
<u>SUBSTRING_INDEX</u>	Extraer subcadenas en función de delimitadores
<u>TRIM</u>	Elimina sufijos y/o prefijos de una cadena.
<u>UCASE</u> o <u>UPPER</u>	Convierte una cadena a mayúsculas
<u>UNCOMPRESS</u>	Descomprime una cadena comprimida mediante <u>COMPRESS</u>
<u>UNCOMPRESSED_LENGTH</u>	Calcula la longitud original de una cadena comprimida
<u>UNHEX</u>	Convierte una cadena que representa un número hexadecimal a cadena de caracteres