http://dev.mysql.com/doc/refman/5.0/es/index.html

MySQL básico

Procedimientos y Funciones

A partir de mysql 5.0, es posible definir rutinas en MySQL. Éstas son funciones y procedimientos que están almacenadas en el servidor de bases de datos y que pueden ser ejecutadas posteriormente. Las funciones (stored functions) regresan un resultado y pueden ser utilizadas en expresiones (de la misma manera en que se usan las funciones de MySQL). Los procedimientos (stored procedures) no regresan ningún valor directamente. Sin embargo, soportan tipos de parámetros cuyo valor puede fijarse en el interior del procedimiento, de tal manera que dicho valor pueda ser utilizado una vez que el procedimiento haya terminado.

Generalmente se usa una función para calcular un valor y regresarlo para utilizarlo posteriormente en alguna expresión. Un procedimiento se usa para producir un efecto o una acción sin necesidad de regresar algún valor. Si se necesita regresar más de un valor, no se puede utilizar una función, sin embargo, se puede crear un procedimiento cuyos parámetros tengan la propiedad de OUT, los cuales puedan ser utilizados en expresiones una vez que el procedimiento se haya ejecutado.

Las ventajas del uso de las rutinas anteriores son las siguientes:

- Extienden la sintaxis de SQL al agregar ciclos e instrucciones de saltos.
- Proveen un mecanismo de manejo de errores.
- Debido a que están almacenadas en el servidor, todo el código necesario para definirlas necesita ser mandado por la red solamente una vez, en el momento de su creación y no cada vez que es invocada. Lo anterior reduce sobrecarga.

Para crear una función o un procedimiento se deben usar las instrucciones CREATE FUNCTION o CREATE PROCEDURE.

Ejemplo:

```
CREATE FUNCTION func1 (date1 DATE, date2 DATE) RETURNS INT
BEGIN
  DECLARE edad INT;
  SET edad = (YEAR(date2) - YEAR(date1));
  RETURN edad;
END
```

Después de haber desarrollado la función anterior, podemos usarla de la siguiente manera:

SELECT func1('1985-12-25', CURDATE());

También se puede utilizar sintaxis parecida a la de pgplsql de la siguiente forma:

```
CREATE FUNCTION `probando`(tipo varchar(25)) RETURNS double
BEGIN
	declare pasta double;
	select sueldo into pasta from gente where nombre=tipo;
	if(pasta is null) then
		return 0;
	else
		return pasta;
	end if;
END
```

Un procedimiento es similar a **una función**, con la diferencia de que no puede regresar ningún valor, así que no incluye ninguna instrucción RETURN. Ejemplo:

```
CREATE PROCEDURE proc1(anio_nacimiento INT)
SELECT nombre, apellido
FROM templa
WHERE YEAR(fechnac) = anio_nacimiento:
```

El resultado del procedimiento anterior no es regresado como valor, sino como **resultset** al cliente que lo manda llamar. Para invocar un procedimiento se debe utilizar la instrucción CALL. Ejemplo:

CALL proc1(1964);

El ejemplo anterior ilustra una cosa que los procedimientos pueden hacer y que las funciones no: **Los procedimientos pueden acceder a tablas**. A su vez, se puede definir un procedimiento que realiza alguna operación determinada en una tabla y que incluya en su definición un parámetro como IN o INOUT para regresar el valor del procedimiento cuando éste regrese. Esta técnica también es utilizada si se necesita regresar más de un valor, ya que una función no puede regresar más de un valor.

De manera predeterminada, un parámetro de un procedimiento es de tipo IN; un parámetro definido de esta manera se recibe en el procedimiento pero cualquier modificación realizada en él no se conservará una vez que el procedimiento termine. Un parámetro OUT es lo contrario: el procedimiento asignará algún valor al parámetro, el cual podrá ser accedido una vez que el procedimiento haya regresado. Un parámetro INOUT permite mandar un valor al procedimiento y obtenerlo de vuelta.

El siguiente ejemplo ilustra lo anterior:

```
CREATE PROCEDURE proc2 (anio_nacimiento INT, OUT cuantos INT)
BEGIN
	DECLARE c CURSOR FOR
	SELECT COUNT(*)
	FROM templa
	WHERE YEAR(fechnac) = anio_nacimiento;
	OPEN c;
```

```
        FETCH c into cuantos;
        CLOSE c;
END;
```

El procedimiento anterior no solamente invoca la instrucción SELECT y asigna el valor de COUNT(*) a una variable. Si hiciera eso, el resultado del query hubiera sido desplegado en la pantalla del cliente. Para evitar ese despliegue, el procedimiento inicializa un cursor y lo usa para ejecutar la instrucción SELECT. De esta manera, el resultado del SELECT se queda dentro del procedimiento de tal manera que pueda procesar el resultado directamente. Para utilizar un cursor, se debe declarar asociándolo al query que será ejecutado. Después se deberá abrir el cursor, obtener los renglones resultantes y por último se deberá cerrar el cursor.

Ahora se puede crear una función que haga uso de este procedimiento para recibir un año y devolver el número de empleados nacidos en ese año

```
CREATE FUNCTION func2(a int) RETURNS int
BEGIN
        declare b int;
        call proc2(a,b);
        return b;
END
```

Por último se puede usar la función mediante ***select func2(1964); → 1***

Estructuras de Control

Las estructuras de control permiten, como su nombre lo indica, controlar el flujo de las instrucciones dentro de un procedimiento o una función. En la siguiente explicación, cada ocurrencia de instrucción(es), indica una lista de una o más instrucciones, cada una de las cuales debe terminar con ";".

Algunas de las estructuras pueden llevar una etiqueta (BEGIN, LOOP, REPEAT y WHILE). Las etiquetas no son sensibles a las mayúsculas o minúsculas pero deben seguir las siguientes reglas:

Si una etiqueta aparece al principio de alguna estructura, deberá también aparecer al final de la misma.

Una etiqueta no deberá aparecer al final de una estructura sin tener su correspondiente pareja al principio de la misma.

BEGIN ... END

BEGIN [instrucción(es)] END

etiqueta: BEGIN [instrucción(es)] END [etiqueta]

La estructura BEGIN … END se utiliza para agrupar un conjunto de instrucciones. Si un procedimiento o una función necesita contener más de una

instrucción, éstas deberán aparecer dentro de un BEGIN … END. De la misma manera, si el procedimiento o función contienen una rutina DECLARE, ésta deberá aparecer al principio del bloque BEGIN … END.

CASE

```
CASE [expresión]
  WHEN expresión1 THEN instruccion(es)
  [WHEN expresión2 THEN instruccion(es)]
  ...
  [ELSE instruccion(es)]
END CASE;
```

IF

```
IF expr1 THEN instruccion(es)
[ELSEIF expr2 THEN instruccion(es)] ...
[ELSE instruccion(es)]
END IF
```

ITERATE

ITERATE etiqueta

ITERATE solamente puede aparecer dentro de un LOOP, REPEAT y WHILE . Lo que realmente significa es: "Haz el ciclo de Nuevo". Por ejemplo:

```
CREATE PROCEDURE doiterate(p1 INT)
BEGIN
        label1: LOOP
        SET p1 = p1 + 1;
        IF p1 < 10 THEN
                ITERATE label1;
        END IF;
        LEAVE label1;
        END LOOP label1;
        SET @x = p1;
END;
```

LEAVE

LEAVE etiqueta

Esta instrucción es utilizada para salir de alguna estructura de control. Puede ser usada dentro de un BEGIN … END o dentro de algún ciclo.

LOOP

```
[etiqueta_inicio:] LOOP
  instruccion(es)
END LOOP [etiqueta_fin]
```

LOOP implementa un ciclo simple, permitiendo que una instrucción o conjunto de instrucciones se repitan. Las instrucciones dentro de este ciclo se repetirán hasta que se ocasione alguna salida, lo cual se hace generalmente con una instrucción LEAVE.

Un ciclo LOOP puede ser etiquetado. etiqueta_fin no puede estar presente a menos que etiqueta_inicio también lo está y, si ambos están presentes, deberán ser iguales.

REPEAT

[etiqueta_inicio:] REPEAT
  instruccion(es)
UNTIL condicion
END REPEAT [etiqueta_fin];

La instrucción o instrucciones dentro de un ciclo REPEAT se repetirán hasta que la condicion sea verdadadera.

Un ciclo REPEAT puede ser etiquetado. etiqueta_fin no puede estar presente a menos que etiqueta_inicio también lo está y, si ambos están presentes, deberán ser iguales.

RETURN

RETURN expresión;

La instrucción RETURN se utiliza solamente dentro de una función. Al ejecutarse, terminará por completo la función dentro de la que se encuentra.

WHILE

[etiqueta_inicio:] WHILE condición DO
    instruccion(es)
END WHILE [etiqueta_fin]

La instrucción o instrucciones dentro de un WHILE serán repetidas mientras la condición sea verdadera.

Un ciclo WHILE puede ser etiquetado. etiqueta_fin no puede estar presente a menos que etiqueta_inicio también lo está y, si ambos están presentes, deberán ser iguales.

Declaraciones

En una declaración se pueden crear:

- Variables locales
- Condiciones
- Cursores
- Manejadores

DECLARE nombre_de_variable [, nombre_de_variable…] TIPO [valor predeterminado];

```
DECLARE nombre_de_condición CONDITION FOR condición

condicion: {SQLSTATE [VALOR] sqlstate_value | mysql_errno}

DECLARE nombre_del_cursor CURSOR FOR instrucción_select

DECLARE handler_type
  HANDLER FOR handler_condition [, handler_condition] …
instruccion

handler_type: {CONTINUE | EXIT}
handler_condition:
{
  SQLSTATE[VALUE] sqlstate_value
  | mysql_errno
  | condition_name
  | SQLWARNING
  | NOT FOUND
  | SQLEXCEPTION
}
```

La declaración de una variable local, una condición, un cursor o un manejador solamente puede aparecer al principio de un bloque BEGIN … END. Si se necesitan hacer diferentes declaraciones, éstas deben hacerse en el siguiente orden:

- Declaración de variables y condiciones
- Declaración de cursores
- Declaración de manejadores

Las variables locales se pueden declarar dentro de alguna rutina en la misma línea (siempre y cuando sean del mismo tipo), separando cada una por una coma. Para darle un valor a éstas o para inicializarlas, se utilzará la instrucción SET.

La instrucción DECLARE … CONDITION crea el nombre para una condición. Dicho nombre puede referirse a una instrucción DECLARE … HANDLER. nombre_de_condición puede ser ya sea un valor SQLSTATE representado por cinco caracteres o un valor numérico específico de MySQL.

La instrucción DECLARE … CURSOR declara un cursor para ser asociado a algún SELECT , el cual no deberá contener la instrucción INTO. El cursor puede abrirse con la cláusula OPEN. Se deberá utilizar la instrucción FETCH para obtener los renglones resultantes del SELECT y se deberá cerrar con la instrucción CLOSE.

La instrucción DECLARE … HANDLER asocia una o más condiciones con una instrucción a ser ejecutada cuando alguna de las condiciones ocurre. El valor del handler_type indica qué ocurre cuando la condición se ejecuta. Con la instrucción CONTINUE , la ejecución de la instrucción continúa, con la instrucción EXIT el bloque BEGIN actual terminará.

handler_condition puede ser alguno de los siguientes valores:

- Un valor de SQLSTATE representado por una cadena de cinco caracteres
- Un valor numérico específico de MySQL
- El nombre de una condición declarada previamente con DECLARE … CONDITION.
- SQLWARNING, el cual cacha cualquier valor de SQLSTATE que empiece con 01.
- NOT FOUND, que cacha cualquier valor de SQLSTATE que empiece con 02.
- SQLEXCEPTION, que cacha cualquier valor de SQLSTATE no cachado por SQLWARNING o NOT FOUND.

Ejemplo:

```
CREATE PROCEDURE ejemplo ()
BEGIN
  DECLARE 'Constraint Violation' CONDITION FOR SQLSTATE '23000';
  DECLARE EXIT HANDLER FOR 'Constraint Violation' ROLLBACK;
  START TRANSACTION;
  INSERT INTO t2 VALUES (1);
  INSERT INTO t2 VALUES (1);
  COMMIT;
END;
```

Instrucciones de Cursores

Los cursores de MySQL son de sólo lectura y pueden ser únicamente utilizados para moverse hacia adelante (hacia el registro siguiente) dentro de un resultset.

OPEN nombre_de_cursor

Abre el cursor para que pueda ser utilizado con la instrucción FETCH.

FETCH [[NEXT] FROM] nombre_de_cursor INTO variable [, variable2,…]

Obtiene el siguiente renglón para el cursor actual y almacena cada una de sus columnas en las variables mencionadas. El cursor debe estar abierto. Si no está disponible ningún renglón, da un error con valor 02000.

CLOSE nombre_de_cursor

Cierra el cursor, el cual deberá estar abierto. Un cursor abierto es cerrado automáticamente cuando el bloque BEGIN dentro del cual está termina.

Comentarios

En las rutinas de MySQL, los comentarios pueden hacerse de tres maneras:

```
# Este es un comentario
/* Este es un comentario de una línea */
/* Este es un comentario
        que abarca
        varias líneas */
-- Este es un comentario
```

Triggers

El soporte para TRIGGERS en MySQL se realizó a partir de la versión 5.0.2. Un trigger puede ser definido para activarse en un INSERT, DELETE o UPDATE en una tabla y puede configurarse para activarse ya sea antes o después de que se haya procesado cada renglón por el query.

Los TRIGGERS en MySQL tienen la misma limitante que las funciones. No pueden referirse a una tabla en general. Pueden solamente referirse a un valor del renglón que está siendo modificado por el query que se está ejecutando.

Las características más importantes de los TRIGGERS son:

- Puede examinar el contenido actual de un renglón antes de que sea borrado o actualizado
- Puede examinar un nuevo valor para ser insertado o para actualizar un renglón de una tabla
- Un BEFORE TRIGGER, puede cambiar el nuevo valor antes de que sea almacenado en la base de datos, lo que permite realizar un filtrado de la información.

El siguiente ejemplo muestra un BEFORE TRIGGER para un SELECT de una tabla:

```
CREATE TABLE t(i INT, dt DATETIME);
CREATE TRIGGER t_ins BEFORE INSERT ON t
        FOR EACH ROW BEGIN
                SET NEW.dt = CURRENT_TIMESTAMP;
                IF NEW.i < 0 THEN
                        SET NEW.i = 0;
                END IF;
END ;
```

Ejemplo de procedure con manejo de CURSOR

codigo	nombre	precio	tipo
1	Pan	0.75	B
2	Leche	1.15	B
3	Yogur	2.16	M
4	Platano	1.75	M
5	Perfume	25.15	A
6	Dentifrico	1.15	A

- Los alimentos B (básicos) bajan un 10%
- Los alimentos M (medios) suben un 5%
- Los alimentos A (normales)
    - Si valen menos de 10 → se quedan con su precio
    - Si valen más de 10 → suben un 50%

```
CREATE DEFINER=`root`@`%` PROCEDURE `actualiza_precios`()
BEGIN
   DECLARE a integer;
   DECLARE b varchar(25);
   DECLARE c double precision;
   DECLARE acabado BOOLEAN DEFAULT FALSE;
   DECLARE salchichon CURSOR FOR SELECT codigo,tipo,precio FROM cosas;
   DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET acabado = TRUE;
   OPEN salchichon;
   REPEAT
   FETCH salchichon INTO a, b, c;
   IF NOT acabado THEN
     IF b='B' THEN
       UPDATE cosas SET precio=0.9*precio WHERE codigo=a;
     ELSE
       IF b='M' THEN
          UPDATE cosas SET precio=1.05*precio WHERE codigo=a;
       ELSE
          IF c>10 THEN
            UPDATE cosas SET precio=1.5*precio WHERE codigo=a;
          END IF;
       END IF;
     END IF;
   END IF;
   UNTIL acabado END REPEAT;
END
```

Call actualiza_precios();

codigo	nombre	precio	tipo
1	Pan	0.675	B
2	Leche	1.035	B
3	Yogur	2.268	M
4	Platano	1.8375	M
5	Perfume	25.15	A
6	Dentifrico	1.15	A