tisBDdocs

Buscar en este sitio

P09 Procedimientos, excepciones y cursores

CONTENIDOS

Crear procedimientos (PROCEDURE)

Los procedimientos o rutinas almacenadas son un conjunto de instrucciones SQL combinadas con una serie de estructuras de control. Se guardan en el servidor, forman parte de una base de datos, y se accede a ellas a través de llamadas.

Sintaxis básica

```
CREATE PROCEDURE nombre_procedimiento ([parámetro[,...]])
BEGIN
  Bloque_de_sentencias
END;
```

Ejemplo de uso:

Procedimento para la puesta a cero de los créditos impartidos.

```
DELIMITER //
CREATE PROCEDURE inicializa_imparte()
BEGIN
  UPDATE imparte set creditos=0;
END;

//
```

Nota: DELIMITER es una instrucción para indicar a mySQL que a partir de ahora, hasta que no introduzcamos // no se acaba la sentencia

Parámetros

Los parámetros se declaran en la cabecera de la instrucción indicando si se trata de un parámetros de entrada (in), salida (out), o ambos (inout)

```
[ IN | OUT | INOUT ] nombre_parámetro tipo_datos
```

Veamos un ejemplo de un procedimiento que toma como entrada un dni de profesor y un código de asignatura y como resultado actualiza el valor de los créditos impartidos por dicho profesor en esa asignatura.

```
DELIMITER //
CREATE PROCEDURE actualiza_creditos(in dniaux varchar(10), in
asigaux char(5))
BEGIN
  UPDATE imparte SET creditos= 3
    WHERE dni = dniaux and asignatura=asigaux;
END;
//
```

Variables

En ocasiones es necesario apoyarse en variables locales para almacenar temporalmente un valor.

Para definir variables se usa la instrucción DECLARE:

```
DECLARE nombre_variable tipo_datos
```

La definición de variables se hace al comenzar el bloque de sentencias a ejecutar (tras BEGIN)

En el siguiente ejemplo se realiza un almacenamiento temporal del valor 'TU' que se va a buscar en la tabla de profesores para proceder a su borrado:

```
DELIMITER //

CREATE PROCEDURE borra_categoria()
BEGIN
    DECLARE auxcat char(4);
    SET auxcat='TU';
    DELETE FROM profesores WHERE categoria=auxcat;
END;
//
```

El siguiente procedimiento se usa para disminuir el número de profesores asociados a una categoría al cambiar de categoría el profesor cuyo dni se pasa por parámetro de entrada. En este caso, la variable la usamos para almacenar el resultado de una consulta previa en la que obtenemos la categoría del profesor. Posteriormente, esa categoría es buscada en la tabla de categorías y se disminuye en 1 el total de profesores asociados a ella.

```
DELIMITER //

CREATE PROCEDURE cambia_de_categoria (in eldni char(9))
BEGIN
  DECLARE auxcat char(4);
  SELECT categoria INTO auxcat
    FROM profesores WHERE dni=eldni;
  UPDATE categoria SET total=total-1 WHERE codigo=auxcat;
END;
//
```

OJO!!! En este caso, la SELECT ha de devolver una fila y tantas columnas como variables tenga para guardar valores a la derecha.

La declaración de una variable admite también la inicialización a un valor por defecto:

```
DECLARE auxcat char(4) DEFAULT '0000';
```

Control de flujo

El procedimiento no deja de ser una pieza de código que en ocasiones requiere la incorporación de instrucciones para control del flujo de código. En este caso se usan estructuras básicas como las que podemos encontrar en cualquier lenguaje de programación: IF, REPEAT, WHILE, LOOP, LEAVE, CASE.

Sentencia **IF:** para indicar alternativas binarias de ejecución dependiendo de ciertas condiciones.

```
IF condición THEN lista_sentencias
```

```
  [ELSEIF condición THEN lista_sentencias] ...
  [ELSE lista_sentencias]
END IF
```

Sentencia **REPEAT:** para reiterar una misma acción hasta cumplir una determinada condición

```
[begin_label:] REPEAT
  lista_sentencias
  UNTIL condición
END REPEAT [end_label]
```

Sentencia **WHILE:** para reiterar una misma acción hasta que se deje de cumplir una determinada condición

```
[begin_label:] WHILE condición DO
  lista_sentencias
END WHILE [end_label]
```

Sentencia **LOOP:** para reiterar una misma acción hasta que se obligue a abandonar el bucle.

```
[begin_label:] LOOP
  lista_sentencias
END LOOP [end_label]
```

Sentencia **LEAVE:** para abandonar cualquier control de flujo etiquetado

```
LEAVE label
```

Sentencia **CASE:** para indicar varias alternativas de ejecución dependiendo de ciertas condiciones.

```
CASE
WHEN condición THEN lista_sentencias
[WHEN condición THEN lista_sentencias] ...
[ELSE lista_sentencias]
```

Manipulación de procedimientos

Una vez que el procedimiento queda almacenado podremos consultar su contenido, borrarlo, modificarlo, y por supuesto, invocarlo para que se ejecute. Las siguientes instrucciones nos permiten la manipulación de procedimientos.

Ver la definición de un procedimiento

```
SHOW CREATE PROCEDURE nombre_procedure;
```

Borrar un procedimiento

```
DROP PROCEDURE nombre_procedure;
```

Llamar a un procedimiento

```
CALL nombre_procedure(par1,par2, ...);
CALL nombre_procedure();
```

Manejo de excepciones (HANDLERS)

Una excepción es el aviso de un error que se está produciendo durante la ejecución de un trozo de código.
En el siguiente ejemplo, si ya hay un profesor con el mismo DNI en la base de datos se producirá un error (clave primaria duplicada) e interrumpirá la ejecución.

```
BEGIN
...
INSERT INTO PROFESORES VALUES ('21456783B', …, ...);
...
END;
```

Si no queremos que se interrumpa la ejecución deberemos capturar el error para decidir qué hacer en ese caso. Para ello deberemos declarar un handler (que podemos traducir literalmente como "manejador" de excepciones) de la siguiente manera:

```
DECLARE tipo_handler HANDLER FOR condición[,...]
lista_sentencias
```

donde **tipo_handler** toma valor de

```
[CONTINUE|EXIT]
```

CONTINUE: la ejecución del programa continua
EXIT: la ejecución termina

y **condición** toma valor de

```
error_code | SQLSTATE [VALUE]
```

error_code y SQLSTATE son valores que los encontramos en el mensaje de error que queremos controlar. Basta con ejecutar el código hasta esperar a que aparezca el mensaje de error, y entonces apuntar dicho código para introducirlo en el manejador, o bien, consultar en la ayuda de MySQL el listado de todos los mensajes de error susceptibles de ser captirados y controlados por los manejadores:
http://dev.mysql.com/doc/refman/5.7/en/error-messages-server.html

```
Error: 1051 SQLSTATE: 42S02 (ER_BAD_TABLE_ERROR)
Message: Unknown table '%s'
```

El siguiente ejemplo capturaría el error de tabla desconocida continuando con la ejecución permitiendo que el resto de instrucciones puedan finalizar:

```
DECLARE CONTINUE HANDLER FOR 1051
BEGIN
-- cuerpo handler
END;
```

El siguiente ejemplo capturaría ese mismo error pero forzando la finalización de la ejecución:

```
DECLARE EXIT HANDLER FOR SQLSTATE '42S02'
BEGIN
 -- cuerpo handler
END;
```

Cursores (CURSOR)

Los cursores son estructuras temporales de almacenamiento auxiliar muy útiles cuando se construyen procedimientos sobre bases de datos. Para crear un cursor es necesario declararlo y definir la consulta select que lo poblará de valores:

```
DECLARE nombre_cursor CURSOR FOR sentencia_select
```

- La sentencia SELECT no puede contener INTO.
- Los cursores no son actualizables.
- Se recorren en un sentido, sin saltos.
- Se deben declarar antes de los handlers y después de la declaración de variables

Ejemplo:

```
DECLARE cur1 CURSOR FOR select dni from profesores;
```

Manipulación de cursores

Para abrir el cursor que se haya definido

```
OPEN nombre_cursor
```

Para desplazarnos por el cursor

```
FETCH nombre_cursor INTO nombre_variable
```

Si no existen más registros disponibles, ocurrirá una condición de Sin Datos con el valor SQLSTATE 02000. Se debe definir una excepción para detectar esta condición

Para cerrar el cursor

```
CLOSE nombre_cursor
```

Ejemplo completo

```
CREATE PROCEDURE asignartipoprofe()
BEGIN
  DECLARE done BOOL DEFAULT 0;
  DECLARE auxdni CHAR(9);
  DECLARE totalt,totalp INT;
  DECLARE cur1 CURSOR FOR SELECT dni FROM profesores;
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;
  OPEN cur1;
    REPEAT
      FETCH cur1 INTO auxdni;
      IF NOT done THEN
        SELECT count(*) into totalt from imparte_teoria where
dni=auxdni;
        SELECT count(*) into totalp from imparte_practicas where
dni=auxdni;
        CASE
          when (totalp+totalt=0) then insert into docente
values(auxdni,'N');
          when (totalp>0 and totalt=0) then insert into docente
values(auxdni,'P');
          when (totalp=0 and totalt>0) then insert into docente
values(auxdni,'T');
          ELSE insert into docente(dni,tipoclase)
values(auxdni,'A');
        END CASE;
      END IF;
    UNTIL done END REPEAT;
  CLOSE cur1;
END;
```

Otras tipos de rutinas almacenadas: funciones (FUNCTION) y disparadores (TRIGGER)

Junto con los **procedimientos almacenados** (PROCEDURE), SQL nos permite crear otras estructuras programables de la misma manera aunque con funcionalidades y naturalezas diferentes. Es el caso de las **funciones** (FUNCTION) y los **disparadores** (TRIGGER).

Una **función es** un procedimiento que obligatoriamente ha de devolver un valor:

```
DELIMITER //
```

```
CREATE FUNCTION total_creditos_profesores()
RETURNS int
BEGIN
  DECLARE total_creditos int;
  SELECT SUM(creditos) INTO total_creditos
  FROM gi_profesores, gi_categorias
  WHERE categoria=codigo;
  RETURN total_creditos;
END
//
```

Un **disparador es** un procedimiento que se ejecuta automáticamente cuando ocurre un determinado evento (inserción, actualización o borrado) sobre una determinada tabla:

```
DELIMITER //
CREATE TRIGGER upd_check BEFORE UPDATE ON account
FOR EACH ROW
BEGIN
  IF NEW.amount < 0 THEN
    SET NEW.amount = 0;
  ELSEIF NEW.amount > 100 THEN
    SET NEW.amount = 100;
  END IF;
END;
//
```

La sintaxis de creación de disparadores responde al siguiente patrón:

```
CREATE TRIGGER nombre_disparador [BEFORE|AFTER]
[INSERT|DELETE|UPDATE]
```

Podrás encontrar más información sobre todo este tema en el Manual de Referencia de MySQL: **http://dev.mysql.com/doc/refman/5.7/en/stored-routines.html**