Procedimientos Almacenados en MySQL,Bucles

Bucles en MySQL

LOOP

Introducción a la sentencia MySQL LOOP

La instrucción LOOP permite ejecutar una o más instrucciones repetidamente.
Aquí está la sintaxis básica de la instrucción LOOP:

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

LOOP puede tener etiquetas opcionales al principio y al final del bloque.
LOOP ejecuta la statement_list repetidamente. La statement_list puede tener una o más declaraciones, cada una terminada por un delimitador de declaración de punto y coma (;).
Por lo general, finaliza el ciclo cuando se cumple una condición utilizando la instrucción LEAVE.
Esta es la sintaxis típica de la instrucción LOOP utilizada con la instrucción LEAVE:

```
[label]: LOOP
    ...
     -- terminate the loop
     IF condition THEN
        LEAVE  [label];
     END IF;
    ...
 END LOOP;
```

La declaración LEAVE sale inmediatamente del bucle. Funciona como la instrucción **break** en otros lenguajes de programación como PHP, C/C++ y Java.
Además de la instrucción LEAVE, puede usar la instrucción ITERATE para omitir la iteración del bucle actual y comenzar una nueva iteración. ITERATE es similar a la instrucción **continue** en PHP, C/C++ y Java.

Ejemplo de sentencia LOOP

La siguiente instrucción crea un procedimiento almacenado que usa una instrucción de bucle LOOP:

```
DROP PROCEDURE LoopDemo;

DELIMITER  $$
CREATE PROCEDUR LoopDemo()
BEGIN
    DECLARE   x    INT ;
    DECLARE  str   VARCHAR (255);

    SET   x  = 1;
    SET  str =   '' ;

   loop_label:   LOOP
        IF x > 10 THEN
            LEAVE loop_label;
        END IF ;

        SET x = x + 1;
        IF (x mod 2) THEN
            ITERATE loop_label;
        ELSE
            SET str = CONCAT (str, x , ',');
        END IF;
    END LOOP;
    SELECT str;
END $$

DELIMITER ;
```

En este ejemplo:

- El procedimiento almacenado construye una cadena a partir de los números pares, por ejemplo, 2, 4 y 6.
- El loop_label antes de la instrucción LOOP para usar con las ITERATE y LEAVE .
- Si el valor de x es mayor que 10 , el ciclo termina debido a la declaración LEAVE .
- Si el valor de la x es un número impar, ITERATE ignora todo lo que está debajo y comienza una nueva iteración de bucle.
- Si el valor de la x es un número par, el bloque en la declaración ELSE construirá la cadena de resultados a partir de números pares.

La siguiente instrucción llama al procedimiento almacenado:

```
CALL  LoopDemo();
```

Aquí está la salida:

```
+-------------+
| str         |
```

```
+-------------+
| 2,4,6,8,10, |
+-------------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.02 sec)
```

Introducción a la declaración de bucle WHILE MySQL

El ciclo WHILE es una instrucción de ciclo que ejecuta un bloque de código repetidamente siempre que una condición sea verdadera.
Aquí está la sintaxis básica de la declaración WHILE :

```
[begin_label:]  WHILE  search_condition  DO
    statement_list
 END WHILE  [end_label]
```

En esta sentencia:

- Primero, especifique una condición de búsqueda después de la palabra clave WHILE .
    - WHILE verifica la search_condition de search_condition al comienzo de cada iteración.
    - Si search_condition evalúa como TRUE , WHILE ejecuta la statement_list siempre que search_condition sea TRUE.

El ciclo WHILE se denomina ciclo de condición previa porque verifica la condición antes de ejecutar la lista de sentencias.

- Segundo, especifique una o más declaraciones que se ejecutarán entre las palabras clave DO y END WHILE .
- Tercero, especifique etiquetas opcionales para la instrucción WHILE al principio y al final de la construcción del bucle.

El siguiente diagrama de flujo ilustra la declaración del bucle WHILE MySQL:

Figura 1: Diagrama de flujo para una sentencia WHILE

Ejemplo de declaración de bucle WHILE

- Primero, cree una tabla llamada calendars que almacene fechas e información de fechas derivadas, como día, mes, trimestre y año:

```
 CREATE TABLE calendars(
    id INT AUTO_INCREMENT,
    fulldate DATE UNIQUE,
    day TINYINT NOT NULL,
    month TINYINT NOT NULL,
    quarter TINYINT NOT NULL,
    year INT NOT NULL,
    PRIMARY KEY (id)
);
```

- En segundo lugar, cree un nuevo procedimiento almacenado para insertar una fecha en la tabla de calendars:

```
DELIMITER  $$

CREATE PROCEDURE InsertCalendar(dt DATE)
BEGIN
    INSERT INTO calendars(
       fulldate,
       day,
       month,
       quarter,
       year
   )
    VALUES(
       dt,
       EXTRACT(DAY FROM dt),
       EXTRACT(MONTH FROM dt),
       EXTRACT(QUARTER FROM dt),
       EXTRACT(YEAR FROM dt)
   );
END $$

DELIMITER  ;
```

- En tercer lugar, cree un nuevo procedimiento almacenado LoadCalendars() que cargue varios días a partir de una fecha de inicio en la tabla de calendars.

```
 DELIMITER  $$

 CREATE PROCEDURE LoadCalendars(
    startDate DATE,
    day INT
)
 BEGIN

     DECLARE counter INT DEFAULT 1;
     DECLARE dt DATE DEFAULT startDate;

     WHILE counter <= day DO
```

```
        CALL InsertCalendar(dt);
        SET counter = counter + 1;
        SET dt = DATE_ADD(dt,INTERVAL 1 day);
    END WHILE;

END $$

DELIMITER ;
```

El procedimiento almacenado LoadCalendars() acepta dos argumentos:
startDate es la fecha de inicio insertada en la tabla de calendars . day es el número de días que se cargarán a partir de startDate .
En el procedimiento almacenado LoadCalendars() :

- Primero, declare un counter y variables dt para mantener valores inmediatos. Los valores predeterminados de counter y dt son 1 y startDate respectivamente.
- Luego, verifique si el counter es menor o igual al day , en caso afirmativo:
    - Llame al procedimiento almacenado InsertCalendar() para insertar una fila en la tabla de calendars .
    - Aumenta el counter en uno. Además, aumente el dt en un día utilizando la función DATE_ADD() .

El ciclo WHILE inserta fechas repetidamente en la tabla de calendars hasta que el counter sea igual al day .
La siguiente instrucción llama al procedimiento almacenado LoadCalendars() para cargar 31 días en la tabla de calendars partir del January 1st 2019 .

Bucle REPEAT

La instrucción REPEAT ejecuta una o más instrucciones hasta que una condición de búsqueda sea verdadera.

Aquí está la sintaxis básica de la declaración de bucle REPEAT :

```
[begin_label:] REPEAT
    statement
 UNTIL search_condition
 END REPEAT [end_label]
```

REPEAT ejecuta la statement hasta que search_condition evalúe como verdadera.
REPEAT verifica la search_condition después de la ejecución de la statement , por lo tanto, la statement siempre se ejecuta al menos una vez. Es por eso que REPEAT también se conoce como un ciclo posterior a la prueba.
La instrucción REPEAT puede tener etiquetas al principio y al final. Estas etiquetas son opcionales.
El siguiente diagrama de flujo ilustra el ciclo REPEAT :

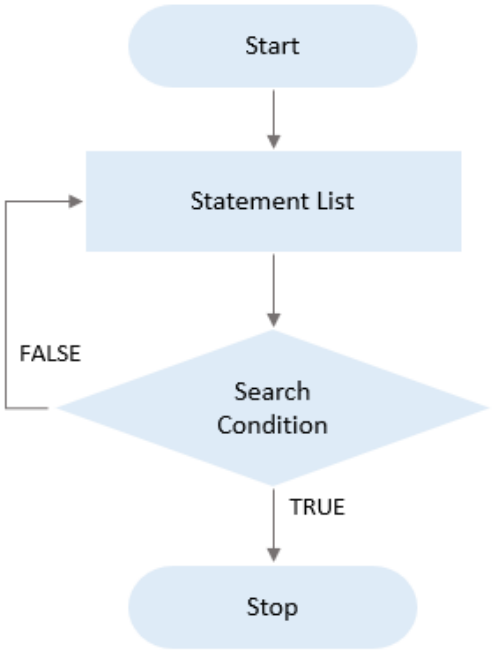

Figura 2: Diagrama de flujo para una sentencia REPEAT

Ejemplo de bucle MySQL REPEAT

Esta instrucción crea un procedimiento almacenado llamado RepeatDemo que usa la instrucción REPEAT para concatenar números del 1 al 9:

```
DELIMITER  $$

CREATE PROCEDURE RepeatDemo()
BEGIN
    DECLARE counter INT DEFAULT 1;
    DECLARE result VARCHAR(100) DEFAULT '';

    REPEAT
        SET result = CONCAT(result,counter,',');
        SET counter = counter + 1;
    UNTIL counter >= 10
    END REPEAT;

    -- display result
    SELECT result;
END $$

DELIMITER ;
```

En este procedimiento almacenado:

- Primero, declare dos variables counter y result y establezca sus valores iniciales en 1 y en blanco. La variable de counter se usa para contar de 1 a 9 en el ciclo. Y la variable de result se usa para almacenar la cadena concatenada después de cada iteración de bucle.

- En segundo lugar, agregua el valor del counter a la variable de result usando la función CONCAT() hasta que el counter sea mayor o igual a 10.

La siguiente instrucción llama al procedimiento almacenado RepeatDemo() :

```
CALL  RepeatDemo();
```

Aquí está la salida:

```
+--------------------+
| result             |
+--------------------+
| 1,2,3,4,5,6,7,8,9, |
+--------------------+
1  row   in   set  (0.02 sec)

Query OK, 0  rows  affected (0.02 sec)
```

Sentencia LEAVE

La instrucción LEAVE mueve el control de flujo hasta una etiqueta dada.
A continuación se muestra la sintaxis básica de la declaración LEAVE:

```
LEAVE  label;
```

En esta sentencia, se especifica la etiqueta del bloque que desea salir después de la palabra clave LEAVE.

Usar la instrucción LEAVE para salir de un procedimiento almacenado

Si [label] es la más externa del procedimiento almacenado o el bloque de funciones, LEAVE termina el procedimiento o la función almacenados.
La siguiente instrucción muestra cómo usar la instrucción LEAVE para salir de un procedimiento almacenado:

```
 CREATE PROCEDURE sp_name()
sp:  BEGIN
     IF condition THEN
         LEAVE sp;
     END IF;
     -- other statement
 END $$
```

Por ejemplo, esta declaración crea un nuevo procedimiento almacenado que verifica el crédito de un cliente determinado en la tabla de customers de la base de datos de muestra:

```
 DELIMITER  $$

 CREATE PROCEDURE CheckCredit(
    inCustomerNumber int
)
```

```
sp:  BEGIN

     DECLARE customerCount INT;

     -- check if the customer exists
     SELECT
         COUNT (*)
     INTO customerCount
     FROM
        customers
     WHERE
        customerNumber = inCustomerNumber;

     -- if the customer does not exist, terminate
     -- the stored procedure
     IF customerCount = 0 THEN
         LEAVE sp;
     END IF ;

     -- other logic
     -- ...
 END $$

 DELIMITER ;
```

Usando la declaración LEAVE en bucles

La declaración LEAVE permite terminar un ciclo. La sintaxis general para la instrucción LEAVE cuando se usa en las instrucciones LOOP, REPEAT y WHILE.
Usando LEAVE con la instrucción LOOP :

```
[label]:  LOOP
     IF condition THEN
         LEAVE [label];
     END IF;
     -- statements
 END LOOP [label];
```

- Usando LEAVE con la declaración REPEAT :

```
[label:] REPEAT
     IF condition THEN
         LEAVE [label];
     END IF ;
     -- statements
 UNTIL search_condition
 END REPEAT [label];
```

- Usando LEAVE con la declaración WHILE :

```
[label:] WHILE search_condition DO
     IF condition THEN
         LEAVE [label];
     END IF;
     -- statements
 END WHILE [label];
```

LEAVE hace que el ciclo actual especificado por [label] finalice. Si un bucle está encerrado dentro de otro bucle, puede romper ambos bucles con una sola declaración LEAVE .

- Ejemplo de uso de LEAVE en un bucle El siguiente procedimiento almacenado genera una cadena de enteros desde el 1 hasta un número aleatorio entre 4 y 10:

```
DELIMITER  $$

CREATE PROCEDURE LeaveDemo(OUT result VARCHAR (100))
BEGIN
    DECLARE counter INT DEFAULT 1;
    DECLARE times INT;
    -- generate a random integer between 4 and 10
    SET times = FLOOR(RAND()*(10 - 4 + 1)+4);
    SET result = '';
   disp: LOOP
        -- concatenate counters into the result
        SET result = concat(result,counter,',');

        -- exit the loop if counter equals times
        IF counter = times THEN
            LEAVE disp;
        END IF ;
        SET counter = counter + 1;
    END LOOP ;
END $$

DELIMITER  ;
```

Esta declaración llama al procedimiento LeaveDemo :

```
CALL LeaveDemo(@result);
SELECT @result;
```

Aquí está uno de los resultados:

```
+------------------+
| @result          |
+------------------+
| 1,2,3,4,5,6,7,8, |
+------------------+
1  row   in   set  (0.00 sec)
```