MySQL®
Conference & Expo

PL/SQL

TRIGGERS
STORED PROCEDURES
FUNCTIONS

Triggers

- ✓ Uma trigger é uma rotina de execução automática, associada a uma tabela
- ✓ A definição de um trigger especifica uma ação a ser executada antes ou após a inserção (insert), atualização (update) ou exclusão (delete) de dados em uma tabela
- ✓ Uma trigger nunca é chamada por uma aplicação ou usuário
- ✓ Triggers são geralmente usadas para validação de dados, cópia de dados entre tabelas, calcular ou preencher o conteúdo de uma coluna derivada de outras colunas e criar logs para registrar a utilização de uma tabela.

Estrutura de Triggers

- Inicia-se com as palavras CREATE TRIGGER seguidas pelo nome do trigger e o nome da tabela à qual o trigger está associado
- Em seguida, define-se o evento que dispara o trigger. As opções são antes (before) ou depois (after) de uma inserção (insert), exclusão (delete) ou atualização (update).
- Se houver variáveis locais, elas devem ser declaradas na seqüência
- Em seguida, comandos SQL, que determinam o comportamento da trigger, devem ser declarados.
- Esses comandos SQL devem estar delimitados por BEGIN-END

Sintaxe dos Triggers

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER} {INSERT | DELETE | UPDATE}
ON table_name
[DECLARE VARIABLE variable_name datatype;]
BEGIN
        statements in PL/SQL
END
```

Exemplo de Trigger

```
DELIMITER  $$


CREATE  TRIGGER  deletar_aluno  BEFORE DELETE
ON Curso
FOR EACH ROW
BEGIN
    DELETE FROM aluno WHERE
codcurso=old.codcurso;
END;
$$


DELIMITER ;
```

```
DELETE FROM curso WHERE codcurso='si'
```

New e Old

- New (vale para insert e update)

  Refere-se ao valor novo de uma coluna após a inserção de uma nova linha ou a atualização de uma linha existente

- Old

  Refere-se ao valor antigo de uma coluna antes da atualização ou deleção

Delimiter

- O operador DELIMITER é usado para mudar o delimitador de comandos, que por padrão é o ";".
- Mudamos o DELIMITADOR para podermos usar o ";" no meio do procedimento. Caso não efetuemos essa troca, o procedimento será enviado pela metade e um erro será enviado ao terminal, por erro na sintaxe.
- DELIMITADOR no MySQL, em outras situações, por padrão também é chamado de terminador.

Outro exemplo de trigger

-- Criando uma trigger para evitar alteracoes erradas de mensalidade, que não pode ser maior que 10%

```
DELIMITER $$
CREATE TRIGGER alterar_curso BEFORE UPDATE ON curso
FOR EACH ROW
BEGIN
    IF (new.mensalidade> 1.1*old.mensalidade) THEN
        SET new.mensalidade=1.1*old.mensalidade;
    END IF;
END;
$$
```

UPDATE curso SET mensalidade=5000 WHERE codcurso='as'

Stored Procedures e Functions

- Stored procedures e funções são rotinas criadas com as instruções **CREATE PROCEDURE** e **CREATE FUNCTION**.
- Um procedimento é chamado usando uma instrução **CALL** e só pode passar valores de retorno usando variáveis de saída(out).
- Funções podem retornar um valor escalar e podem ser chamadas de dentro de uma instrução como qualquer outra função (isto é, chamando o nome da função).
- Rotinas armazenadas podem chamar outras rotinas armazenadas. Uma rotina pode ser tanto um procedimento como uma função.

Stored Procedures

- Uma stored procedure é composta de um cabeçalho e um corpo.

- O cabeçalho de uma procedure contém:
  - O nome da stored procedure, que tem que ser único entre nome de tabelas e procedures do BD
  - Uma lista de parâmetros e seus tipos de dados, que a procedure recebe do programa chamador

- O corpo de uma procedure body contém:
  - Uma lista opcional de variáveis locais e seus tipos de dados
  - Um bloco de declarações deleimitadas por BEGIN e END. Um bloco pode conter outros blocos aninhados.

Os Parâmetros de uma Procedure

- *proc_name*: seu procedimento armazenado deve ter um nome, para quando for chamado, podermos então usá-lo;
- *tipo_param*: existem 3 tipos de parâmetros em uma Stored Procedure no MySQL:
- IN => este é um parâmetro de entrada, ou seja, um parâmetro cujo seu valor será utilizado no interior do procedimento para produzir algum resultado;
- OUT => esté parâmetro retorna algo de dentro do procedimento para o lado externo, colocando os valores manipulados disponíveis na memória ou no conjunto de resultados;
- INOUT => faz os dois trabalhos ao mesmo tempo!

Create Procedure

[CREATE PROCEDURE <NOME_PROCEDURE> ([<p1>,...,<pN>])

BEGIN
<instruções>;

CREATE PROCEDURE procedure_name

(IN | OUT | INOUT variable_name datatype)

BEGIN

    statements in PL/SQL

END

Exemplo de Procedure

```
DELIMITER |

CREATE PROCEDURE contaluno (OUT result INT)
BEGIN
  SELECT COUNT(*) INTO result FROM ALUNO;
END
|
```

```
Sql> CALL contaluno(@TOTAL)
Sql> SELECT @TOTAL
```

7

Create Function

- CREATE FUNCTION function_name
- (parameter)
- BEGIN
- statements in PL/SQL
- END

A instrução CREATE FUNCTION é usada em versão novas do MySQL para suporte a UDFs (User Defined Functions - Funções Definidas pelo Usuário). As UDFs continuam a ser suportadas, mesmo com a existencia de stored functions. Uma UDF pode ser considerada como uma stored function externa.

Exemplo de Function

- DELIMITER |
- CREATE FUNCTION hello (s CHAR(20))
- RETURNS CHAR(50)
- RETURN CONCAT('Hello, ',s,'!');
- |

```
Sql>SELECT HELLO(´BIANCA´);
```

Hello, BIANCA!

Comando de seleção IF

```
IF condição THEN
        <comando1>;
        <comando2>;
END IF;
```

```
IF condição1 THEN
        <comando1>;
ELSEIF condição2 THEN
        <comando2>;
      ELSE
        <comando3>;
END IF;
```

```
IF SAL < 2000 THEN
        update emp
        set SAL = SAL * 1.2;
ELSEIF SAL < 3000
        update emp
        set SAL = SAL * 1.1;
ELSE
        update emp
        set SAL = SAL * 1.05;
END IF;
```

Comando de seleção CASE

```
CASE VARIÁVEL
        WHEN VALOR1 THEN COMANDO1;
        WHEN VALOR2 THEN COMANDO2;
        ELSE COMANDO3;
END;
```

```
CASE SEXO
        WHEN ‘F’ THEN ‘FEMININO’
        WHEN ‘M’ THEN ‘MASCULINO’
END;
```

Comando de repetição LOOP

```
LOOP
    comandos ;
END LOOP;
```

```
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
```

Comando de repetição REPEAT

[begin_label:] REPEAT

comandos;

UNTIL condição

END REPEAT [end_label]

```
CREATE PROCEDURE dorepeat(p1 INT)
BEGIN
        SET @x = 0;
        REPEAT
                SET @x = @x + 1;
        UNTIL @x > p1
        END REPEAT;
 END
|
```

Comando de repetição WHILE

[begin_label:] WHILE condição DO
comandos;
END WHILE [end_label]

```
CREATE PROCEDURE dowhile()
BEGIN
    DECLARE v1 INT DEFAULT 5;
     WHILE v1 > 0 DO
         ...
         SET v1 = v1 - 1;
     END WHILE;
END
```

Cursores

- Mantêm o resultado de uma consulta
- Somente podem ser lidos (*read-only*)
- Podem ser percorridos apenas sequencialmente (*non-scrolling*)

Operações com Cursores

- Declaração de cursor
- DECLARE cursor_name CURSOR FOR sql_statement
- Abertura de cursor
  - OPEN cursor_name
- Percorrendo o cursor
  - FETCH cursor_name INTO variables
- Fechamento do cursor
  - CLOSE cursor_name

Exemplo

```
DELIMITER $$
CREATE PROCEDURE copialunas( )
BEGIN
DECLARE a,b CHAR(9);
DECLARE c    CHAR(30);
DECLARE d   CHAR(3);
DECLARE e    ENUM('feminino','masculino');
DECLARE fim INT DEFAULT 0;
DECLARE cur1 CURSOR FOR SELECT * FROM aluno WHERE sexo='feminino';
DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET fim = 1;
-- Usando cursor
OPEN cur1;
REPEAT
 FETCH cur1 INTO a, b,c,d,e;
 INSERT INTO ALUNAS(RA,RG,NOME,CODCURSO,sexo) VALUES(a,b,c,d,e);
UNTIL fim END REPEAT;
CLOSE cur1;
END
$$
```

Handlers

- Certas condições podem exigir tratamento específico. Estas condições podem estar relacionadas a erros, bem como controle de fluxo.
  - DECLARE handler_type HANDLER FOR condition_value[,...] sp_statement
- Onde:

handler_type: CONTINUE | EXIT

condition_value: SQLSTATE [VALUE] sqlstate_value | condition_name | SQLWARNING | NOT FOUND | SQLEXCEPTION | mysql_error_code

Exemplo: copiar alunas

```
DELIMITER $$
CREATE PROCEDURE copialunas( )
BEGIN
DECLARE a,b CHAR(9);
DECLARE c    CHAR(30);
DECLARE d   CHAR(3);
DECLARE e    ENUM('feminino','masculino');
DECLARE fim INT DEFAULT 0;
DECLARE cur1 CURSOR FOR SELECT * FROM aluno WHERE sexo='feminino';
DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET fim = 1;
-- Usando cursor
OPEN cur1;
REPEAT
 FETCH cur1 INTO a, b,c,d,e;
 INSERT INTO ALUNAS(RA,RG,NOME,CODCURSO,sexo) VALUES(a,b,c,d,e);
UNTIL fim END REPEAT;
CLOSE cur1;
END
$$
```