STORED ROUTINES

Wagner Bianchi
Certified MySQL 5.0 Developer
Certified MySQL 5.0 Database Administrator
Certified MySQL 5.1 Cluster Database Administrator

Introdução

- O que será apresentado:
  - Comentários;
  - User Variables;
  - Prepared Statements;
  - Stored Routines;
    - Stored Procedures;
    - Stored Functions;
    - Stored Cursors.

Comentários

- Existem alguas formas de comentários no MySQL, geralmente utilizado em meio aos procedimentos que veremos neste módulo:

```
-- esse é um comentário de linha;

#  esse também é um comentário de linha;

/* esse é um comentário de bloco */
```

User Variables

- User variable são escritas como **@var_name**, e aceitam valores inteiros, reais, string ou **NULL**;
- User variables ou variáveis do usuário são variáveis que podem ser utilizadas para armazenar um valor para exibir ou mesmo para que o seu valor seja utilizado mais tarde, durante a rotina de um programa;
- O valor de uma User variable pode ser setado através da declaração **SET**, assim como:

```
SET @var_name = 'Wagner Bianchi';
```

- Utilizam o @ precedente ao seu nome;
- Apresentam particularidades na sua sintaxe quando colocadas em meio a um **SELECT** e em meio a uma rotina;

User Variables

- Se o valor de uma user variable não for setado em meio a uma **SELECT** ou mesmo uma routine, seu valor será **NULL**;

```
mysql> select @var_name;
+------------+
| @var_name  |
+------------+
| NULL       |
+------------+
1 row in set (0.00 sec)
```

User Variables

- Nomes de user variables não são case-sensitive, tango faz **`@Var_Name`** como **`@var_name`**. As duas variáveis são a mesma;

```
mysql> SELECT @var_name;
+-----------------+
| @var_name       |
+-----------------+
| Wagner Bianchi |
+-----------------+
1 row in set (0.00 sec)

mysql> SELECT @var_name, @Var_Name, @VAR_NAME;
+-----------------+-----------------+-----------------+
| @var_name       | @Var_Name       | @VAR_NAME       |
+-----------------+-----------------+-----------------+
| Wagner Bianchi | Wagner Bianchi | Wagner Bianchi |
+-----------------+-----------------+-----------------+
1 row in set (0.00 sec)
```

User Variables

- Em meio a **SELECT**, temos que utilizar o sinal de igualdade no formato *Pascal, como segue:

```
mysql> SELECT @var_nome:=nome FROM exemplo;
+------------------+
| @var_nome:=nome  |
+------------------+
| Wagner Bianchi   |
+------------------+
1 row in set (0.09 sec)
```

- User variables são bastante utilizadas com Prepared Statements;

Exercícios

- Com base no assunto apresentado “**user variables**”, responda a **LISTA 5** de exercícios.

- **Fonte de referência**: http://dev.mysql.com/doc/refman/5.6/en/user-variables.html

Prepared Statements

- Os *Prepared Statements* ou mesmo, declarações preparadas, é um tipo de recurso que a linguagem SQL no MySQL implementa para que se escreva menos e se tenha maior rapidez na execução de consultas que requerem pequenas modificações;

- Podemos preparar uma declaração qualquer e executá-la por várias vezes, com vários valores. Apresenta mais performace devido ao comando ou declaração ser *parseado* apenas uma vez;

- Possibilidades de resultados já armazenados em memória;

- Menos tráfego entre o servidor e o cliente, menos coversões de tipos - cada conexão pode preparar as suas declarações, sem visilidade à outras conexões de outros usuários;

Prepared Statements

- Um exemplo de como inicializar um *Prepared Statements*:

```
mysql> PREPARE stmt FROM 'SELECT NOW()';
Query OK, 0 rows affected (0.08 sec)
Statement prepared
```

- Com a declaração **EXCUTE**, executamos o procedimento armazenado no SGBD MySQL;

```
mysql> EXECUTE stmt;
+---------------------+
| NOW()               |
+---------------------+
| 2008-08-12 15:36:18 |
+---------------------+
1 row in set (0.00 sec)
```

Prepared Statements

- Podemos implementar declarações mais sofisticadas, utilizando o banco de dados world:

```
mysql> PREPARE stmt FROM
  -> 'SELECT b.Name, a.Language
  '> FROM CountryLanguage AS a INNER JOIN Country AS b
  '> ON a.CountryCode = b.Code
  '> WHERE a.Language =?';
Query OK, 0 rows affected (0.00 sec)
Statement prepared
```

- A consulta que foi preparada, retorna quais países fala uma determinada língua. Para encaixar um valor sobre o ponto de interrogação, utilizaremos uma user variable com o valor *Portuguese*;

Prepared Statements

- Executando um *Prepared Statement* com auxílio de user variables:

```
mysql> SET @var ='Portuguese'; EXECUTE stmt USING @var;
Query OK, 0 rows affected (0.00 sec)

+---------------+------------+
| Name          | Language   |
+---------------+------------+
| Andorra       | Portuguese |
| Brazil        | Portuguese |
| Portugal      | Portuguese |
| …             |            |
| France        | Portuguese |
| United States | Portuguese |
+---------------+------------+
12 rows in set (0.02 sec)
```

Prepared Statements

- Utilizamos **`USING`** para indicar a variável para o *Prepared Statement*. O valor dessa variável substitui o ponto de interrogação e a consulta então é executada;

- A consulta é executada várias vezes, somente com variações na condição da cláusula **`WHERE`**;

- Podemos utilizar mais de uma variável para combinar com mais de um ponto de interrogação em um *Prepared Statement*. A ordem das variáveis deve seguir a ordem de encaixe na consulta armazenada. Para utilizar mais de uma variável na cláusula **`USING`**, separe-as com vírgula;

- Para apagar uma declaração preparada do servidor MySQL, utilize **`DEALLOCATE PREPARE`** *name_prepared_statement* ou ainda

  `DROP PREPARE` *name_prepared_statement*;

Exercícios

- Com base no assunto apresentado "***Prepared Statements***", responda a **LISTA 6** de exercícios.

O mascote chamado Sakila e o MySQL Proxy, utilizado para balanceamento de carga no MySQL.

- **Fonte de referência**: http://dev.mysql.com/doc/refman/5.0/en/user-variables.html

Stored Procedures

- Um Stored Procedure ou um procedimento armazenado é uma rotina ou programa que fica armazenado dentro de um banco de dados;

- Possui
  - Um nome único
  - Uma lista de parâmetros
  - Um valor ou não de retorno
  - Um conjunto de comandos SQL

Leitura complementar:

- http://imasters.uol.com.br/artigo/7556/mysql/stored_procedures_no_mysql/

Stored Procedures

- Principais benfícios de utilizar Stored procedures:
  - Stored Procedures tendem a ser mais rápidas
    - Ficam armazenadas no servidor;
    - Se beneficiam de caches e buffers do sistema;
    - Não necessita trafegar comandos para serem executados;
    - Reduz o tempo de parser;
  - São componentes
    - A lógica do negócio está codificada e armazenada na banco;
    - Não há problema de mudar a linguagem de programação do sistema;
  - São portáveis
    - Podem executar em qualquer ambiente que o SGBD rodar;
    - O que pode não ser verdadeiro para algumas linguagens de programação

Stored Procedures

- Um procedure simples:

```
mysql> CREATE PROCEDURE sp1() SELECT * FROM Country LIMIT 1;
Query OK, 0 rows affected (0.00 sec)
```

- Uma vez definido o procedimento, podemos invocá-lo com a declaração **CALL** *nome_proc*();

```
mysql> CALL sp1();
+--------+
| Name   |
+--------+
| Brazil |
+--------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.01 sec)
```

Stored Procedures

- Um procedimento ou Stored Procedure pode receber parâmetros, o que torna os seus resultados mais **dinâmicos**;

- O parâmetros em um Stored Procedure podem ser dos seguintes tipos:
  - **IN**: parâmtero de entrada, o valor deste tipos de parâmtero somente tem valor internamente ao procedimento. Uma vez que o procedimento chega ao fim, este valor é destruído;

  - **OUT**: este valor deverá ser passado através de uma user variable, sendo que o valor desta variável será setado internamente ao procedimento, caso isso não aconteça, o valor da user variable será **NULL**. Após o encerramento do procedimento, o valor da variável é colocado em memória e pode ser acessado com um **SELECT** `@var`;

  - **INOUT**: esse tipo de parâmetro nos permitirá fazer as duas coisas, enviar um valor que não será ignorado, sendo preciso setar um novo valor para que este esteja disponível após o ecerramento do procedimento.

Stored Procedures

- Podemos criar procedimentos com múltiplos comandos. Para isso, precisamos delimiter o corpo da rotina com **BEGIN/END**;

```
mysql> CREATE PROCEDURE sp2(IN v_name CHAR(80))
    -> BEGIN
    ->  SELECT v_name AS par1;
    ->  SELECT v_name AS par2;
    ->  SELECT v_name AS par3;
    -> END;
```

- Entre **BEGIN/END** podemos também declarar *local variables* – diferente de *user variables* para trabalhar valores internamente aos procedimentos;

Stored Procedures

- Cada *local variable* declarada dentro de um bloco devem ter nomes diferentes, sendo que cada uma é local de cada bloco e somente serão acessíveis dentro do seu bloco;

- Esta restrição também se aplica à declaração de condições e *cursors*;

- Para declarar uma variável local, utilize **`DECLARE`**, prosseguido do nome da variável mais o seu tipo de dado, que ainda pode receber a cláusula **`DEFAULT`**, que seta a seu valor automaticamente caso a variável não receba um valor. Uma variável com o valor **`NULL`** pode afetar todo o processamento de um procedimento;

- Para variáveis declaradas sem a cláusula **`DEFAULT`**, seu valor inicial é **`NULL`**;

Stored Procedures

- Uma variável local pode ter um valor setado através do cláusula SET, como a seguir:

```
BEGIN
   DECLARE var_nome INT DEFAULT 0;
   SET var_nome = 'Wagner Bianchi'
   SELECT var_nome AS nome;
END;
```

- Podemos também, recuperar um valor armazenado em uma coluna de uma tabela qualquer para dentro de uma variável local com **SELECT ... INTO**, como a seguir:

```
BEGIN
   DECLARE var_nome INT DEFAULT 0;
   SELECT Name INTO var_nome FROM Country WHERE Code ='BRA';
END;
```

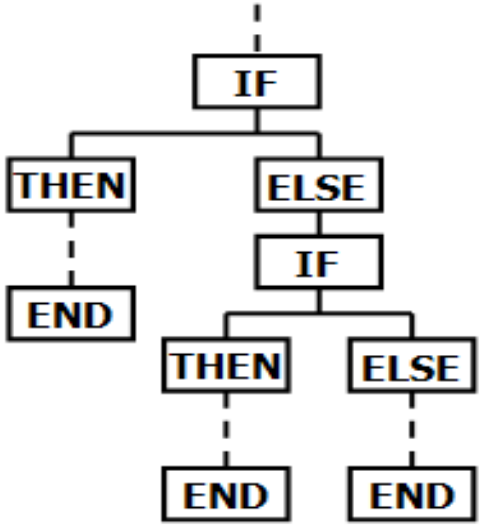
Stored Procedures

- Dentro de procedimentos que utilizam-se de *Cursors* para recuperar dados, se pode utilizar a declaração **FETCH ... INTO**, como segue:

```
BEGIN
      DECLARE v_code CHAR(80);
      DECLARE row_count INT DEFAULT 0;
      DECLARE cur1 CURSOR FOR SELECT Code FROM Country;
      OPEN cur1;
      BEGIN
         DECLARE EXIT HANDLER FOR SQLSTATE '02000' BEGIN END;
         LOOP
           FETCH cur1 INTO v_code;
           SET row_count = row_count + 1;
         END LOOP;
      END;
      CLOSE cur1;
      SELECT CONCAT('Existem ', row_count, ' paises cadastrados!');
END;
```

Stored Procedures

- Estrturas condicionais são utilizadas para mudar o fluxo de execução de um programa ou procedure;
- Permitem utilizar expressões lógicas para definir qual o caminho a ser seguido pelo programa;
- Existem duas estruturas condicionais:
  - **IF-THEN-ELSE**
  - **CASE**



Stored Procedures

```
IF var > 0 THEN
     SELECT 'maior que 0' AS Msg;
ELSE
     SELECT 'menor ou igual a 0' Msg;
END IF;
```

… ou ainda:

```
CASE var
     WHEN var > 0 THEN 'maior que 0'
     WHEN var < 0 THEN 'menor que 0'
   ELSE 'igual a 0'
END CASE;
```

Stored Procedures

- Estruturas de repetição permitem executar um mesmo bloco de código várias vezes;
- Utiliza estruturas condicionais para determinar o fim da iteração;
- Existem basicamente 3 construções:
  - **WHILE**
  - **REPEAT**
  - **LOOP**
- Controle do Loop:
  - **ITERATE**
  - **LEAVE**

Stored Procedures

- A construção **WHILE**:
  - Sempre avalia a condição e, enquanto retornar **TRUE**, continua iterando. Nesse exemplo, passamos dois números e equanto num_1 for menor ou igual a num_2, fazemos **INSERT** na tabela t:

```
CREATE PROCEDURE sp_while(IN num_1 INT, IN num_2 INT)
BEGIN
     IF num_1 < num_2 THEN
           WHILE num_1 <= num_2 DO
                    INSERT INTO t SET id =num_1;
                    SET num_1 = num_1 + 1;
            END WHILE;
         ELSE
            SELECT 'Número menor que 0!' AS Msg;
     END IF;
END;
```

Stored Procedure

- A construção **REPEAT**:
  - Cria um loop condicional, avaliado com uma estrutura condicional **IF-THEN-ELSE**:

```
CREATE PROCEDURE sp_repeat(IN num INT)
BEGIN
     DECLARE i INT DEFAULT 0;
        REPEAT
          SET i = i + 1;
          SELECT i;
          UNTIL i > num
     END REPEAT;
 END;
```

Stored Procedure

- A construção **LOOP** pode ser utiliza livremente ou ainda, podemos definir um label para controlar o mesmo utilizando o controle de loop **LEAVE**;

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

Stored Procedure

- As estruturas **LEAVE** e **ITERATE** permitem desviar a execução de um loop;
- Necessário o uso dos labels que são alvos para desvios de fluxo;
- **LEAVE**
  - Interrompe a execução do loop e a rotina continua a partir do primeiro comando após o loop;
- **ITERATE**
  - Retorna a execução para o início do loop, utilizado para pular uma iteração do comando e começar no próximo passo;

Stored Procedures

```
CREATE PROCEDURE it_leave()
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);
         SET v = v + 1;
         IF v >= 5 THEN
            LEAVE loop_label;
         END IF;
    END LOOP;
END;
```

Stored Procedures

Demonstrações

- Um valor `NULL` no meio do processamento;
- Criando um procedimento simples;
- Criando procedimentos explorando tipos de parâmetros;
- Criando um procedimento com características de segurança;
- Criando um procedimento com múltiplos comandos;
- Criando um procedimento com tratamento de erro;

Exercícios

- Com base no assunto apresentado "***Stored Procedures***", resolva a **LISTA 7** de exercícios.

- **Fonte de referência**: http://dev.mysql.com/doc/refman/5.6/en/stored-routines.html

Functions

- A principal diferença entre uma ***Function*** e um ***Stored Procedure*** é que ela sempre retornará um valor, enquanto que um procedimento, pode retornar ou não um valor;

- *Functions* também estão diretamente vinculadas à um database e podem ser utilizadas em meio a outras ***Functions***, ***View***, ***Trigger*** ou mesmo ***Stored Procedures***;

- Quanto aos metadados, uma Functions fica armazenada na mesma tabela do dicionário de dados onde ficam os Stored Procedures.

```
mysql> SELECT ROUTINE_NAME, ROUTINE_TYPE
    -> FROM INFORMATION_SCHEMA.ROUTINES
    -> WHERE ROUTINE_NAME ='fu1'\G
*************************** 1. row ***************************
ROUTINE_NAME: fu1
ROUTINE_TYPE: FUNCTION
1 row in set (0.00 sec)
```

Functions

- A sintaxe para criação de uma função é bem parecida com aquela apresentada para criação de um ***Stored Procedure***;

```
CREATE FUNCTION nome_func (par 1, par n…)
RETURNS data_type
BEGIN
     CORPO DA ROTINA ou Routine Body
END;
```

- Podemos utilizar tanto *user variables* quanto *variáveis locais* em meio às funções.

Functions

- Validando um e-mail com uma ***Function***:

```
mysql> create function fu1(par char(80))
    -> returns tinyint
    -> begin
    ->   declare var char(120); -- local variable
    ->   set var = locate('@', par);
    ->   begin
    ->     if var <> 0 then
    ->        set @str = 1; -- user variable
    ->     else
    ->        set @str = 0; -- user variable
    ->     end if;
    ->     return @str; -- retorno
    ->   end;
    -> end;
    -> //
Query OK, 0 rows affected (0.00 sec)
```

Functions

```
mysql> SELECT IF(fu1(’me@wagnerbianchi.com') = 1,
       'Email válido!', 'E-mail inválido!') AS TesteMail;
+---------------+
| TesteMail     |
+---------------+
| Email válido! |
+---------------+
1 row in set (0.00 sec)

mysql> SELECT IF(fu1('wagnerbianchi') = 1, 'Email válido!',
       'E-mail inválido!') AS TesteMail;
+------------------+
| TesteMail        |
+------------------+
| E-mail inválido! |
+------------------+
1 row in set (0.00 sec)
```

Functions

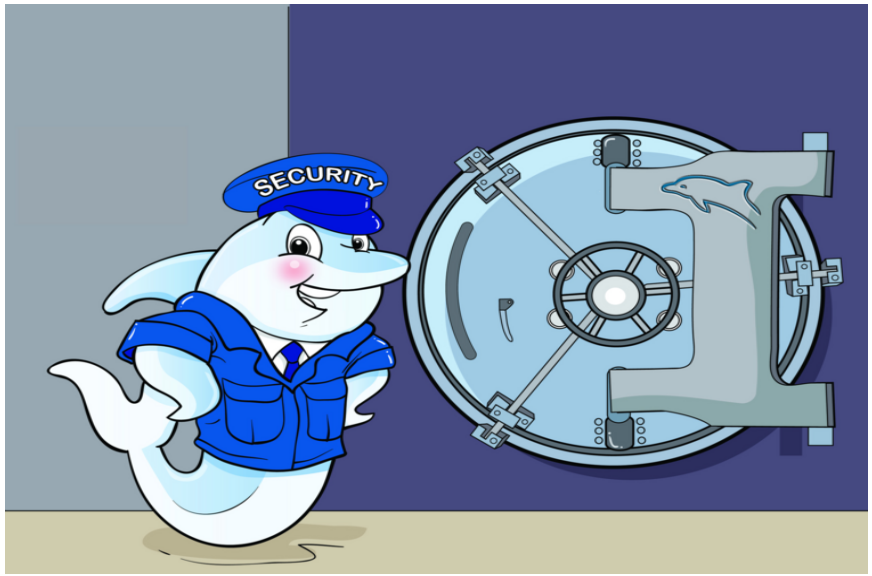
- O comando **`ALTER FUNCTION`** somente alterará as cláuslas:
  - **`DETERMINISTIC`** or **`NOT DETERMINISTIC`**

    Se uma função é determinística, ela sempre produz o mesmo resultado, caso contrário, ela não será determinística.
  - **`SQL SECURITY { INVOKER | DEFINER }`**

    Determina com quais privilégios a função será executada, de quem o invoca ou de quem o criou.
- Não altera a lógica da função.

Functions

- Com base no assunto apresentado “***Functions***”, resolva a **LISTA 8** de exercícios.



- **Fonte de referência**: http://dev.mysql.com/doc/refman/5.6/en/create-function-udf.html