CpSc 462/662: Database Management Systems (DBMS) (TEXNH Approach)

Stored Procedure

James Wang

Stored procedures or functions

- **Stored routines (procedures and functions) are supported in MySQL 5.0.**
  - A stored procedure is a set of SQL statements that can be stored in the server.
  - With stored procedures, clients don't need to keep reissuing the individual statements. Instead a simple call to the stored procedure will perform the same functions.
  - Stored routines can provide improved performance because less information exchanged between the server and the client.

Create Stored Routines

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

CREATE
    [DEFINER = { user | CURRENT_USER }]
    FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body

proc_parameter:
        [ IN | OUT | INOUT ] param_name type
func_parameter:
        param_name type
```


Create Stored Routines (cont.)

type:

Any valid MySQL data type

characteristic:

```
    LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA |
    MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'
```

routine_body:

Valid SQL procedure statement

Parameters

- **Parameters are declared in the parentheses and they can be any valid data types. A function may have no parameter declared.**
- **Each parameter is an IN parameter by default. You can also explicitly specify a parameter to be OUT or INOUT parameter if you define a PROCEDURE (not FUNCTION).**
  - An IN parameter passes a value into a procedure. The procedure might modify the value, but the modification is not visible to the caller when the procedure returns.
  - An OUT parameter passes a value from the procedure back to the caller. Its initial value is NULL within the procedure, and its value is visible to the caller when the procedure returns.
  - An INOUT parameter is initialized by the caller, can be modified by the procedure, and any change made by the procedure is visible to the caller when the procedure returns.
- **Note**
  - Specifying a parameter as IN, OUT, or INOUT is valid only for a PROCEDURE. For FUNCTION, parameters are all IN.

Example

```
mysql> delimiter //

mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)
  -> BEGIN
  ->   SELECT COUNT(*) INTO param1 FROM t;
  -> END;
  -> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> CALL simpleproc(@a);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @a;
+------+
| @a   |
+------+
| 3    |
+------+
1 row in set (0.00 sec)
```

Delimiters

- The previous example uses the mysql client delimiter command to change the statement delimiter from ; to // while the procedure is being defined.
- This allows the ; delimiter used in the procedure body to be passed through to the server rather than being interpreted by mysql itself.
- When using the delimiter command, you should avoid the use of the backslash ("\") character because that is the escape character for MySQL.
- It is unnecessary to use delimiter command when the function definition contains no internal ; statement delimiters.

7

Function Returns

- It is mandatory that a FUNCTION includes the RETURNS clause, which indicates the return type of the function.
- The function body must contain a RETURN value statement.
- If the RETURN statement returns a value of a different type, the value is coerced to the proper type.
  - For example, if a function specifies an ENUM or SET value in the RETURNS clause, but the RETURN statement returns an integer, the value returned from the function is the string for the corresponding ENUM member of set of SET members.

8

Routine Characteristics

- Several characteristics provide information about the nature of data use by the routine.
  - In MySQL, these characteristics are advisory only. The server does not use them to constrain what kinds of statements a routine will be allowed to execute.
  - **CONTAINS SQL:** the routine does not contain statements that read or write data (this is the default characteristic).
  - **NO SQL:** the routine contains no SQL statements.
  - **READS SQL DATA:** the routine contains statements that read data (for example, SELECT), but not statements that write data.
  - **MODIFIES SQL DATA:** the routine contains statements that may write data (for example, INSERT or DELETE).
  - The **SQL SECURITY** characteristic can be used to specify whether the routine should be executed using the permissions of the user who creates the routine or the user who invokes it.

9

Alter or Drop Routines

```
ALTER {PROCEDURE | FUNCTION} sp_name
    [characteristic ...]

characteristic:
    { CONTAINS SQL | NO SQL | READS SQL DATA |
    MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'
```

- This statement can be used to change the characteristics of a stored procedure or function.

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

- This statement is used to drop a stored procedure or function.

10

Call Procedures or Functions

CALL ***sp_name***([***parameter***[,...]])
CALL ***sp_name***[()]

- The CALL statement invokes a procedure that was defined previously with CREATE PROCEDURE.
- CALL can pass back values to its caller using parameters that are declared as OUT or INOUT parameters.
- It also "returns" the number of rows affected, which a client program can obtain at the SQL level by calling the ROW_COUNT() function and from other programming language by calling the mysql_affected_rows() API function.

11

Procedure Call Example

```
CREATE PROCEDURE p (OUT ver_param VARCHAR(25), INOUT incr_param INT)
BEGIN
  # Set value of OUT parameter
  SELECT VERSION() INTO ver_param;
  # Increment value of INOUT parameter
  SET incr_param = incr_param + 1;
END;

mysql> SET @increment = 10;
mysql> CALL p(@version, @increment);
mysql> SELECT @version, @increment;
+------------+------------+
| @version   | @increment |
+------------+------------+
| 5.0.25-log | 11         |
+------------+------------+
```

12

BEGIN ... END

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

- **BEGIN ... END syntax is used for writing compound statements, which can appear within stored routines and triggers.**
- **statement_list represents a list of one or more statements. Each statement within statement_list must be terminated by a semicolon (;) statement delimiter.**
- **Use of multiple statements requires that a client is able to send statement strings containing the ; statement delimiter.**
  - This is handled in the mysql command-line client by using the delimiter command to change the delimiter from ; to other symbols (e.g. //).

13

Declare variables in routines

```
DECLARE var_name[,...] type [DEFAULT value]
```

- **This statement is used to declare local variables.**
- **To provide a default value for the variable, include a DEFAULT clause.**
- **The value can be specified as an expression; it need not be a constant.**
- **If the DEFAULT clause is missing, the initial value is NULL.**
- **Local variables are treated like routine parameters with respect to data type and overflow checking.**
- **The scope of a local variable is within the BEGIN ... END block where it is declared. The variable can be referred to in blocks nested within the declaring block, except those blocks that declare a variable with the same name.**

14

Assign values to variables

```
SET var_name = expr [, var_name = expr] ...
SELECT col_name[,...] INTO var_name[,...] table_expr
```

- **The SET statement in stored routines is an extended version of the general SET statement. Referenced variables may be ones declared inside a routine, or global system variables.**
- **This SELECT syntax stores selected columns directly into variables. Therefore, only a single row may be retrieved.**

```
SELECT id,data INTO x,y FROM test.t1 LIMIT 1;
```

- **Note:**
  - SQL variable names should not be the same as column names.
  - If an SQL statement, such as a **SELECT ... INTO** statement, contains a reference to a column and a declared local variable with the same name, MySQL currently interprets the reference as the name of a variable.

15

Flow Control - IF

```
IF search_condition THEN statement_list
   [ELSEIF search_condition THEN statement_list] ...
   [ELSE statement_list]
END IF
```

- **IF implements a basic conditional construct. If the *search_condition* evaluates to true, the corresponding SQL statement list is executed.**
- **If no *search_condition* matches, the statement list in the ELSE clause is executed. Each *statement_list* consists of one or more statements**

16

Flow Control - CASE

```
CASE case_value
   WHEN when_value THEN statement_list
   [WHEN when_value THEN statement_list] ...
   [ELSE statement_list]
END CASE
```

```
CASE
   WHEN search_condition THEN statement_list
   [WHEN search_condition THEN statement_list] ...
   [ELSE statement_list]
END CASE
```

- **The CASE statement for stored routines implements a complex conditional construct. If a *search_condition* evaluates to true, the corresponding SQL statement list is executed.**
- **If no search condition matches, the statement list in the ELSE clause is executed.**
- **Each *statement_list* consists of one or more statements.**

17

Flow Control - LOOP

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

- **LOOP implements a simple loop construct, enabling repeated execution of the statement list, which consists of one or more statements.**
- **The statements within the loop are repeated until the loop is exited; usually this is accomplished with a LEAVE statement.**
- **A LOOP statement can be labeled.**
  - ***end_label*** cannot be given unless ***begin_label*** also is present. If both are present, they must be the same.

18

Flow Control - REPEAT

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

- **The statement list within a REPEAT statement is repeated until the search_condition is true.**
  - REPEAT always enters the loop at least once.
  - statement_list consists of one or more statements.
- **A REPEAT statement can be labeled.**
  - end_label cannot be given unless begin_label also is present. If both are present, they must be the same.

19

Flow Control - WHILE

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

- **The statement list within a WHILE statement is repeated as long as the search_condition is true.**
- **statement_list consists of one or more statements.**
- **A WHILE statement can be labeled.**
  - end_label cannot be given unless begin_label also is present. If both are present, they must be the same.

20

Flow Control – LEAVE or ITERATE

LEAVE ***label***

- **This statement is used to exit any labeled flow control construct.**
- **It can be used within BEGIN ... END or loop constructs (LOOP, REPEAT, WHILE).**

ITERATE ***label***

- **ITERATE can appear only within LOOP, REPEAT, and WHILE statements.**
- **ITERATE means "do the loop again."**

21

Conditions

```
DECLARE condition_name CONDITION FOR
    condition_value

condition_value:
    SQLSTATE [VALUE] sqlstate_value
  | mysql_error_code
```

- **This statement specifies conditions that need specific handling. It associates a name with a specified error condition.**
  - The name can subsequently be used in a DECLARE HANDLER statement.
- **A *condition_value* can be an SQLSTATE value or a MySQL error code.**

22

Handlers

```
DECLARE handler_type HANDLER FOR condition_value[,...] statement

handler_type:
    CONTINUE
  | EXIT
  | UNDO

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

- The **DECLARE ... HANDLER** statement specifies handlers that each may deal with one or more conditions. If one of these conditions occurs, the specified *statement* is executed.

23

Handler Types

- **For a CONTINUE handler, execution of the current routine continues after execution of the handler statement.**
- **For an EXIT handler, execution terminates for the BEGIN ... END compound statement in which the handler is declared. (This is true even if the condition occurs in an inner block.)**
- **The UNDO handler type statement is not yet supported.**
- **If a condition occurs for which no handler has been declared, the default action is EXIT.**

24



References

- An Introduction to Database Systems, Eighth Edition, C. J. Date, Addison Wesley, 2004, ISBN: 0-321-19784-4.
- Database Management Systems, Third Edition, Raghu Ramakrishnan and Johannes Gehrke, McGraw-Hill, 2002, ISBN: 0072465638.
- http://dev.mysql.com/doc/refman/5.1/en/stored-procedures.html

