



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.





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.





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.



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



DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name

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





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.



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; FND:

mysql> SET @increment = 10;
mysql> CALL p(@version, @increment);
mysql> SELECT @version, @increment

| 5.0.25-log | 11





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
- 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. //).



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
- 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



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;

- - SQL variable names should not be the same as column
 - # 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.



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





Flow Control - CASE

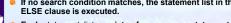
CASE case_value WHEN when_value THEN statement_list [WHEN when value THEN statement list] .. [ELSE statement_list] END CASE

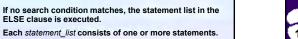
CASE

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







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

