

Stored Functions

This lesson is about creating, viewing and deleting stored functions. We also discuss the key differences between stored procedures and stored functions.

We'll cover the following ^

- Syntax

Stored Functions

A stored function is a kind of a stored program that can only return one value. It can be used in SQL statements in place of an expression. Common formulas or expressions that are used over and over again can be written in stored functions to make the code reusable. However, if a stored function that queries data from tables is used in a SQL statement, then it may slow down the speed of the query.

There are a number of differences when we compare stored functions to stored procedures. Stored procedures can call stored functions but the opposite is not possible. Stored functions can be used in SQL statements but stored procedures can only be called with the **CALL** keyword. That is why stored procedures are stored in compiled form where as stored functions are parsed and compiled at runtime. Return value is optional in stored procedures but a must in stored functions. Moreover, stored functions can only return one value but there is no such restriction on the number of return values in stored procedures. Stored functions only support **IN** parameter type while stored procedures can have **IN**, **OUT** and **INOUT** parameters. Error handling is not possible in stored functions.

The **CREATE FUNCTION** statement is used to create a stored function. The parameter list contains all the parameters of the function. Unlike stored

parameter list contains all the parameters of the function. Unlike stored procedures where the parameters could be **IN**, **OUT** or **INOUT** type, a stored function only takes **IN** parameters so there is no need to specify the type of parameters in the parameter list. Since the stored function can return only one value, the data type of the return value is specified after the **RETURN** keyword.

A stored function can be deterministic or non deterministic meaning that for the same input parameters, the result will either be the same or different. This can be specified by using the keywords **DETERMINISTIC** or **NOT DETERMINISTIC**. If this keyword is not specified, the type is set to **NOT DETERMINISTIC** by default. The function body must have at least one **RETURN** statement. When control reaches it, the stored function exits.

To view all functions in a database, **SHOW FUNCTION STATUS** statement is used. This results returned can be narrowed down based on the **LIKE** operator or any other condition specified in the optional **WHERE** clause. To delete a stored function, **DROP FUNCTION** keywords with the optional **IF EXISTS** clause is used.

Syntax

```
DELIMITER **
```

```
CREATE FUNCTION function_name(parameter_list)
```

```
RETURNS datatype
```

```
[NOT] DETERMINISTIC
```

```
BEGIN
```

```
function body
```

```
END **
```

```
DELIMITER ;
```

```
SHOW FUNCTION STATUS [LIKE 'pattern' | WHERE condition];
```

DROP FUNCTION [IF EXISTS] **function_name**;

Connect to the terminal below by clicking in the widget. Once connected, the command line prompt will show up. Enter or copy-paste the command `./DataJek/Lessons/59lesson.sh` and wait for the mysql prompt to start-up.

-- The lesson queries are reproduced below for convenient copy/paste into the terminal.



-- Query 1

DELIMITER **

```
CREATE FUNCTION DigitalAssetCount(  
    ID INT)
```

```
RETURNS VARCHAR(50)
```

```
DETERMINISTIC
```

```
BEGIN
```

```
    DECLARE ReturnMessage VARCHAR(50);
```

```
    DECLARE Number INT DEFAULT 0;
```

```
    SELECT COUNT(*) INTO Number FROM DigitalAssets WHERE ActorId = ID;
```

```
    IF Number = 0 THEN
```

```
        SET ReturnMessage = 'The Actor does not have any digital assets.';
```

```
    ELSE
```

```
        SET ReturnMessage = CONCAT('The Actor has ', Number, ' digital assets');
```

```
    END IF;
```

```
    -- return the customer level
```

```
    RETURN (ReturnMessage);
```

```
END**
```

```
DELIMITER ;
```

-- Query 2

```
SHOW FUNCTION STATUS;
```

-- Query 3

```
SHOW FUNCTION STATUS
```

```
WHERE db = 'MovieIndustry';
```

-- Query 4

```
SHOW FUNCTION STATUS
```

```
LIKE '%Count%';
```

-- Query 5

```
SELECT Id, DigitalAssetCount(Id) AS Count
```

```
FROM Actors;
```

-- Query 6

```
DELIMITER **
```

```
CREATE PROCEDURE GetDigitalAssetCount(  
    IN ActorNo INT,
```

```
    OUT Message VARCHAR(50))
```

```

BEGIN
    DECLARE Number INT DEFAULT 0;
    SET Number = ActorNo;

    SET Message = DigitalAssetCount(Number);
END**
DELIMITER ;

-- Query 7
CALL GetDigitalAssetCount(10,@assetcount);
SELECT @assetcount;

-- Query 8
DELIMITER **
CREATE FUNCTION TimeSinceLastUpdate(
    ID INT,
    Asset VARCHAR(15))
RETURNS INT
NOT DETERMINISTIC
BEGIN
    DECLARE ElapsedTime INT;
    SELECT TIMESTAMPDIFFF(SECOND, LastUpdatedOn, NOW())
    INTO ElapsedTime
    FROM DigitalAssets
    WHERE ActorID = ID AND AssetType = Asset;
    RETURN ElapsedTime;
END**
DELIMITER ;

-- Query 9
SELECT TimeSinceLastUpdate(1,'Instagram');

-- Query 10
SELECT routine_name
FROM information_schema.routines
WHERE routine_type = 'FUNCTION' AND routine_schema = 'MovieIndustry';

-- Query 11
DROP FUNCTION DigitalAssetCount;
DROP FUNCTION IF EXISTS DigitalAssetCount;
SHOW WARNINGS;

```

Terminal



1. We will create a stored function to count the number of digital assets owned by an actor and return this number to the caller.

```

DELIMITER **

CREATE FUNCTION DigitalAssetCount(
    ID INT)
RETURNS VARCHAR(50)
DETERMINISTIC
BEGIN

```

```

DECLARE ReturnMessage VARCHAR(50);
DECLARE Number INT DEFAULT 0;

SELECT COUNT(*) INTO Number FROM DigitalAssets WHERE ActorId = ID
;

IF Number = 0 THEN
    SET ReturnMessage = 'The Actor does not have any digital asse
ts.';
ELSE
    SET ReturnMessage = CONCAT('The Actor has ', Number, ' digita
l assets');
END IF;

-- return the customer level
RETURN (ReturnMessage);
END**
DELIMITER ;

```

2. To view the functions in a database, the **SHOW FUNCTION STATUS** statement is used.

```
SHOW FUNCTION STATUS;
```

Output of the above statement is captured in the widget below:

SHOW FUNCTION STATUS OUTPUT

```
mysql> SHOW FUNCTION STATUS;
```

Db	Name	Type	Definer	Modified
MovieIndustry	DigitalAssetCount	FUNCTION	root@localhost	2020-06-02 04
sys	extract_schema_from_file_name	FUNCTION	mysql.sys@localhost	2020-06-02 04

Description

Takes a raw file path, and attempts to extract the schema name from it.

Useful for when interacting with Performance Schema data concerning IO statistics, for example.

Currently relies on the fact that a table data file will be within a specified database directory (will not work with partitions or tables that specify an individual DATA_DIRECTORY).

Parameters

path (VARCHAR(512)):

The full file path to a data file to extract the schema name from.

Returns

VARCHAR(64)

Example

```
mysql> SELECT sys.extract_schema_from_file_name('/var/lib/mysql/employees/employee.ibd');
+-----+
| sys.extract_schema_from_file_name('/var/lib/mysql/employees/employee.ibd') |
+-----+
| employees                                                                    |
+-----+
1 row in set (0.00 sec)
| utf8          | utf8_general_ci      | utf8_general_ci      |
| sys          | extract_table_from_file_name | FUNCTION | mysql.sys@localhost | 2020-06-02 04
Description
```

Takes a raw file path, and extracts the table name from it.

Useful for when interacting with Performance Schema data concerning IO statistics, for example.

Parameters

path (VARCHAR(512)):

The full file path to a data file to extract the table name from.

Returns

VARCHAR(64)

Example

```
mysql> SELECT sys.extract_table_from_file_name('/var/lib/mysql/employees/employee.ibd');
+-----+
| sys.extract_table_from_file_name('/var/lib/mysql/employees/employee.ibd') |
+-----+
| employee                                                                    |
+-----+
1 row in set (0.02 sec)
| sys          | format_bytes          | FUNCTION | mysql.sys@localhost | 2020-06-02 04
Description
```

Takes a raw bytes value, and converts it to a human readable format.

Parameters

bytes (TEXT):

A raw bytes value.

Returns

TEXT

Example

```
mysql> SELECT sys.format_bytes(2348723492723746) AS size;
+-----+
| size      |
+-----+
```

```
| 2.09 PiB |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT sys.format_bytes(2348723492723) AS size;
+-----+
| size      |
+-----+
| 2.14 TiB  |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT sys.format_bytes(23487234) AS size;
+-----+
| size      |
+-----+
| 22.40 MiB |
+-----+
1 row in set (0.00 sec)
```

sys	format_path	FUNCTION	mysql.sys@localhost	2020-06-02 04
Description				

Takes a raw path value, and strips out the datadir or tmpdir replacing with @@datadir and @@tmpdir respectively.

Also normalizes the paths across operating systems, so backslashes on Windows are converted to forward slashes

Parameters

path (VARCHAR(512)):
The raw file path value to format.

Returns

VARCHAR(512) CHARSET UTF8

Example

```
mysql> select @@datadir;
+-----+
| @@datadir |
+-----+
| /Users/mark/sandboxes/SmallTree/AMaster/data/ |
+-----+
1 row in set (0.06 sec)
```

```
mysql> select format_path('/Users/mark/sandboxes/SmallTree/AMaster/data/mysql/proc.MYD') AS path;
+-----+
| path |
+-----+
| @@datadir/mysql/proc.MYD |
+-----+
1 row in set (0.03 sec)
```

sys	format_statement	FUNCTION	mysql.sys@localhost	2020-06-02 04
Description				

Formats a normalized statement, truncating it if it is > 64 characters long by default.

To configure the length to truncate the statement to by default, update the `statement truncate l

variable with `sys_config` table to a different value. Alternatively, to change it just for just your particular session, use `SET @sys.statement_truncate_len := <some new value>`.

Useful for printing statement related data from Performance Schema from the command line.

Parameters

statement (LONGTEXT):
The statement to format.

Returns

LONGTEXT

Example

```
mysql> SELECT sys.format_statement(digest_text)
-> FROM performance_schema.events_statements_summary_by_digest
-> ORDER by sum_timer_wait DESC limit 5;
+-----+
| sys.format_statement(digest_text) |
+-----+
| CREATE SQL SECURITY INVOKER VI ... KE ? AND `variable_value` > ? |
| CREATE SQL SECURITY INVOKER VI ... ait` IS NOT NULL , `esc` . ... |
| CREATE SQL SECURITY INVOKER VI ... ait` IS NOT NULL , `sys` . ... |
| CREATE SQL SECURITY INVOKER VI ... , `compressed_size` ) ) DESC |
| CREATE SQL SECURITY INVOKER VI ... LIKE ? ORDER BY `timer_start` |
+-----+
5 rows in set (0.00 sec)
| utf8 | utf8_general_ci | utf8_general_ci |
| sys | format_time | FUNCTION | mysql.sys@localhost | 2020-06-02 04
Description
```

Takes a raw picoseconds value, and converts it to a human readable form.

Picoseconds are the precision that all latency values are printed in within Performance Schema, however are not user friendly when wanting to scan output from the command line.

Parameters

picoseconds (TEXT):
The raw picoseconds value to convert.

Returns

TEXT

Example

```
mysql> select format_time(342342342342345);
+-----+
| format_time(342342342342345) |
+-----+
| 00:05:42 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> select format_time(342342342);
+-----+
| format time(342342342) |
```



```

+-----+
| 342.34 us |
+-----+

```

1 row in set (0.00 sec)

```
mysql> select format_time(34234);
```

```

+-----+
| format_time(34234) |
+-----+
| 34.23 ns |
+-----+

```

1 row in set (0.00 sec)

	utf8	utf8_general_ci	utf8_general_ci	
sys	list_add	FUNCTION	mysql.sys@localhost	2020-06-02 04:00:00

Description

Takes a list, and a value to add to the list, and returns the resulting list.

Useful for altering certain session variables, like sql_mode or optimizer_switch for instance.

Parameters

in_list (TEXT):

The comma separated list to add a value to

in_add_value (TEXT):

The value to add to the input list

Returns

TEXT

Example

```
mysql> select @@sql_mode;
```

```

+-----+
| @@sql_mode |
+-----+
| ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION |
+-----+

```

1 row in set (0.00 sec)

```
mysql> set sql_mode = sys.list_add(@@sql_mode, 'ANSI_QUOTES');
```

Query OK, 0 rows affected (0.06 sec)

```
mysql> select @@sql_mode;
```

```

+-----+
| @@sql_mode |
+-----+
| ANSI_QUOTES,ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION |
+-----+

```

1 row in set (0.00 sec)

	utf8	utf8_general_ci	utf8_general_ci	
sys	list_drop	FUNCTION	mysql.sys@localhost	2020-06-02 04:00:00

Description

Takes a list, and a value to attempt to remove from the list, and returns the resulting list.

Useful for altering certain session variables, like sql_mode or optimizer_switch for instance.

Parameters

in_list (TEXT):

The comma separated list to drop a value from

in_drop_value (TEXT):

The value to drop from the input list

Returns

TEXT

Example

```
mysql> select @@sql_mode;
```

```
+-----+
| @@sql_mode |
+-----+
| ANSI_QUOTES,ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION |
+-----+
1 row in set (0.00 sec)
```

```
mysql> set sql_mode = sys.list_drop(@@sql_mode, 'ONLY_FULL_GROUP_BY');
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> select @@sql_mode;
```

```
+-----+
| @@sql_mode |
+-----+
| ANSI_QUOTES,STRICT_TRANS_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION |
+-----+
1 row in set (0.00 sec)
```

sys	ps_is_account_enabled	FUNCTION	mysql.sys@localhost	2020-06-02 04
utf8	utf8_general_ci	utf8_general_ci		

Description

Determines whether instrumentation of an account is enabled within Performance Schema.

Parameters

in_host VARCHAR(60):

The hostname of the account to check.

in_user VARCHAR(32):

The username of the account to check.

Returns

ENUM('YES', 'NO', 'PARTIAL')

Example

```
mysql> SELECT sys.ps_is_account_enabled('localhost', 'root');
```

```
+-----+
| sys.ps_is_account_enabled('localhost', 'root') |
+-----+
| YES |
+-----+
1 row in set (0.01 sec)
```

sys	ps_is_consumer_enabled	FUNCTION	mysql.sys@localhost	2020-06-02 04

Description

Determines whether a consumer is enabled (taking the consumer hierarchy into consideration) within the Performance Schema.

An exception with errno 3047 is thrown if an unknown consumer name is passed to the function. A consumer name of NULL returns NULL.

Parameters

in_consumer VARCHAR(64):
The name of the consumer to check.

Returns

ENUM('YES', 'NO')

Example

```
mysql> SELECT sys.ps_is_consumer_enabled('events_stages_history');
+-----+
| sys.ps_is_consumer_enabled('events_stages_history') |
+-----+
| NO                                                    |
+-----+
1 row in set (0.00 sec)
```

sys	ps_is_instrument_default_enabled	FUNCTION	mysql.sys@localhost	2020-06-02 04:04:04
-----	----------------------------------	----------	---------------------	---------------------

Description

Returns whether an instrument is enabled by default in this version of MySQL.

Parameters

in_instrument VARCHAR(128):
The instrument to check.

Returns

ENUM('YES', 'NO')

Example

```
mysql> SELECT sys.ps_is_instrument_default_enabled('statement/sql/select');
+-----+
| sys.ps_is_instrument_default_enabled('statement/sql/select') |
+-----+
| YES                                                            |
+-----+
1 row in set (0.00 sec)
```

sys	ps_is_instrument_default_timed	FUNCTION	mysql.sys@localhost	2020-06-02 04:04:04
-----	--------------------------------	----------	---------------------	---------------------

Description

Returns whether an instrument is timed by default in this version of MySQL.

Parameters

in_instrument VARCHAR(128):
The instrument to check.

Returns

ENUM('YES', 'NO')

Example

```
mysql> SELECT sys.ps_is_instrument_default_timed('statement/sql/select');
+-----+
| sys.ps_is_instrument_default_timed('statement/sql/select') |
+-----+
| YES                                                         |
+-----+
1 row in set (0.00 sec)
```

sys	ps_is_thread_instrumented	FUNCTION	mysql.sys@localhost	2020-06-02 04:04:04
-----	---------------------------	----------	---------------------	---------------------

Description

Checks whether the provided connection id is instrumented within Performance Schema.

Parameters

in_connection_id (BIGINT UNSIGNED):
The id of the connection to check.

Returns

ENUM('YES', 'NO', 'UNKNOWN')

Example

```
mysql> SELECT sys.ps_is_thread_instrumented(CONNECTION_ID());
+-----+
| sys.ps_is_thread_instrumented(CONNECTION_ID()) |
+-----+
| YES                                             |
+-----+
```

sys	ps_thread_account	FUNCTION	mysql.sys@localhost	2020-06-02 04:04:04
-----	-------------------	----------	---------------------	---------------------

Description

Return the user@host account for the given Performance Schema thread id.

Parameters

in_thread_id (BIGINT UNSIGNED):
The id of the thread to return the account for.

Example

```
mysql> select thread_id, processlist_user, processlist_host from performance_schema.threads where
+-----+-----+-----+
| thread_id | processlist_user | processlist_host |
+-----+-----+-----+
| 23        | NULL             | NULL             |
| 30        | root             | localhost        |
| 31        | msandbox         | localhost        |
| 32        | msandbox         | localhost        |
+-----+-----+-----+
4 rows in set (0.00 sec)
```

```
mysql> select sys.ps_thread_account(31);
+-----+
| sys.ps_thread_account(31) |
+-----+
```

msandbox@localhost					
+-----+					
1 row in set (0.00 sec)					
	utf8	utf8_general_ci	utf8_general_ci		
sys	ps_thread_id	FUNCTION	mysql.sys@localhost	2020-06-02 04	
Description					

Return the Performance Schema THREAD_ID for the specified connection ID.

Parameters

in_connection_id (BIGINT UNSIGNED):

The id of the connection to return the thread id for. If NULL, the current connection thread id is returned.

Example

```
mysql> SELECT sys.ps_thread_id(79);
```

```
+-----+
| sys.ps_thread_id(79) |
+-----+
|                98 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT sys.ps_thread_id(CONNECTION_ID());
```

```
+-----+
| sys.ps_thread_id(CONNECTION_ID()) |
+-----+
|                98 |
+-----+
1 row in set (0.00 sec)
```

sys	ps_thread_stack	FUNCTION	mysql.sys@localhost	2020-06-02 04
Description				

Outputs a JSON formatted stack of all statements, stages and events within Performance Schema for the specified thread.

Parameters

thd_id (BIGINT UNSIGNED):

The id of the thread to trace. This should match the thread_id column from the performance_schema.threads table.

in_verbose (BOOLEAN):

Include file:lineno information in the events.

Example

(line separation added for output)

```
mysql> SELECT sys.ps_thread_stack(37, FALSE) AS thread_stack\G
```

```
***** 1. row *****
thread_stack: {"rankdir": "LR", "nodesep": "0.10", "stack_created": "2014-02-19 13:39:03",
"mysql_version": "5.7.3-m13", "mysql_user": "root@localhost", "events":
[{"nesting_event_id": "0", "event_id": "10", "timer_wait": 256.35, "event_info":
"sql/select", "wait_info": "select @@version_comment limit 1\nerrors: 0\nwarnings: 0\nlock time:
...

```

sys	ps_thread_trx_info	FUNCTION	mysql.sys@localhost	2020-06-02 04
Description				

Returns a JSON object with info on the given threads current transaction, and the statements it has already executed, derived from the `performance_schema.events_transactions_current` and

`performance_schema.events_statements_history` tables (so the consumers for these also have to be enabled within Performance Schema to get full data in the object).

When the output exceeds the default truncation length (65535), a JSON error object is returned, such as:

```
{ "error": "Trx info truncated: Row 6 was cut by GROUP_CONCAT()" }
```

Similar error objects are returned for other warnings/and exceptions raised when calling the function.

The max length of the output of this function can be controlled with the `ps_thread_trx_info.max_length` variable set via `sys_config`, or the `@sys.ps_thread_trx_info.max_length` user variable, as appropriate.

Parameters

`in_thread_id` (BIGINT UNSIGNED):

The id of the thread to return the transaction info for.

Example

```
SELECT sys.ps_thread_trx_info(48)\G
***** 1. row *****
```

```
sys.ps_thread_trx_info(48): [
{
  "time": "790.70 us",
  "state": "COMMITTED",
  "mode": "READ WRITE",
  "autocommitted": "NO",
  "gtid": "AUTOMATIC",
  "isolation": "REPEATABLE READ",
  "statements_executed": [
  {
    "sql_text": "INSERT INTO info VALUES (1, 'foo')",
    "time": "471.02 us",
    "schema": "trx",
    "rows_examined": 0,
    "rows_affected": 1,
    "rows_sent": 0,
    "tmp_tables": 0,
    "tmp_disk_tables": 0,
    "sort_rows": 0,
    "sort_merge_passes": 0
  },
  {
    "sql_text": "COMMIT",
    "time": "254.42 us",
    "schema": "trx",
    "rows_examined": 0,
    "rows_affected": 0,
    "rows_sent": 0,
    "tmp_tables": 0,
    "tmp_disk_tables": 0,
    "sort_rows": 0,
    "sort_merge_passes": 0
  }
]
]
```

```

},
{
  "time": "426.20 us",
  "state": "COMMITTED",
  "mode": "READ WRITE",
  "autocommitted": "NO",
  "gtid": "AUTOMATIC",
  "isolation": "REPEATABLE READ",
  "statements_executed": [
    {
      "sql_text": "INSERT INTO info VALUES (2, 'bar')",
      "time": "107.33 us",
      "schema": "trx",
      "rows_examined": 0,
      "rows_affected": 1,
      "rows_sent": 0,
      "tmp_tables": 0,
      "tmp_disk_tables": 0,
      "sort_rows": 0,
      "sort_merge_passes": 0
    },
    {
      "sql_text": "COMMIT",
      "time": "213.23 us",
      "schema": "trx",
      "rows_examined": 0,
      "rows_affected": 0,
      "rows_sent": 0,
      "tmp_tables": 0,
      "tmp_disk_tables": 0,
      "sort_rows": 0,
      "sort_merge_passes": 0
    }
  ]
}
]
}
]
1 row in set (0.03 sec)
| utf8 | utf8_general_ci | utf8_general_ci |
| sys | quote_identifier | FUNCTION | mysql.sys@localhost | 2020-06-02 04:04:04
Description

```

Takes an unquoted identifier (schema name, table name, etc.) and returns the identifier quoted with backticks.

Parameters

in_identifier (TEXT):
The identifier to quote.

Returns

TEXT

Example

```

mysql> SELECT sys.quote_identifier('my_identifier') AS Identifier;
+-----+
| Identifier |
+-----+
| `my_identifier` |
+-----+
1 row in set (0.00 sec)

```

```
mysql> SELECT sys.quote_identifer('my`idenfier') AS Identifier;
```

```
+-----+
```

```
| Identifier |
```

```
+-----+
```

```
| `my``idenfier` |
```

```
+-----+
```

```
1 row in set (0.00 sec)
```

```
| sys | sys_get_config | FUNCTION | mysql.sys@localhost | 2020-06-02 04:04:04 |
Description
```

Returns the value for the requested variable using the following logic:

1. If the option exists in sys.sys_config return the value from there.
2. Else fall back on the provided default value.

Notes for using sys_get_config():

* If the default value argument to sys_get_config() is NULL and case 2. is reached, NULL is returned. It is then expected that the caller is able to handle NULL for the given configuration option.

* The convention is to name the user variables @sys.<name of variable>. It is <name of variable> that is stored in the sys_config table and is what is expected as the argument to sys_get_config().

* If you want to check whether the configuration option has already been set and if not assign with the return value of sys_get_config() you can use IFNULL(...) (see example below). However this should not be done inside a loop (e.g. for each row in a result set) as for repeated calls where assignment is only needed in the first iteration using IFNULL(...) is expected to be significantly slower than using an IF (...) THEN ... END IF; block (see example below).

Parameters

in_variable_name (VARCHAR(128)):

The name of the config option to return the value for.

in_default_value (VARCHAR(128)):

The default value to return if the variable does not exist in sys.sys_config.

Returns

VARCHAR(128)

Example

```
mysql> SELECT sys.sys_get_config('statement_truncate_len', 128) AS Value;
```

```
+-----+
```

```
| Value |
```

```
+-----+
```

```
| 64 |
```

```
+-----+
```

```
1 row in set (0.00 sec)
```

```
mysql> SET @sys.statement_truncate_len = IFNULL(@sys.statement_truncate_len, sys.sys_get_config('statement_truncate_len', 64));
Query OK, 0 rows affected (0.00 sec)
```

```
IF (@sys.statement_truncate_len IS NULL) THEN
```

```
SET @sys.statement_truncate_len = sys.sys_get_config('statement_truncate_len', 64);
```

```
END IF;
```

```
| utf8 | utf8_general_ci | utf8_general_ci |
```

```
| sys | version_major | FUNCTION | mysql.sys@localhost | 2020-06-02 04:04:04 |
Description
```

Returns the major version of MySQL Server.

Returns

TINYINT UNSIGNED

Example

```
mysql> SELECT VERSION(), sys.version_major();
```

```
+-----+-----+
| VERSION() | sys.version_major() |
+-----+-----+
| 5.7.9-enterprise-commercial-advanced | 5 |
+-----+-----+
1 row in set (0.00 sec)
```

sys	version_minor	FUNCTION	mysql.sys@localhost	2020-06-02 08:00:00
Description				

Returns the minor (release series) version of MySQL Server.

Returns

TINYINT UNSIGNED

Example

```
mysql> SELECT VERSION(), sys.server_minor();
```

```
+-----+-----+
| VERSION() | sys.version_minor() |
+-----+-----+
| 5.7.9-enterprise-commercial-advanced | 7 |
+-----+-----+
1 row in set (0.00 sec)
```

sys	version_patch	FUNCTION	mysql.sys@localhost	2020-06-02 08:00:00
Description				

Returns the patch release version of MySQL Server.

Returns

TINYINT UNSIGNED

Example

```
mysql> SELECT VERSION(), sys.version_patch();
```

```
+-----+-----+
| VERSION() | sys.version_patch() |
+-----+-----+
| 5.7.9-enterprise-commercial-advanced | 9 |
+-----+-----+
1 row in set (0.00 sec)
```

```
+-----+-----+-----+-----+
23 rows in set (0.01 sec)
```

2. The above statement only shows those functions from the database

currently used that the user has privilege to view. We can specify search condition in **WHERE** clause as follows:

```
SHOW FUNCTION STATUS
WHERE db = 'MovieIndustry';
```

Similarly **LIKE** operator can also be used to narrow down the search results:

```
SHOW FUNCTION STATUS
LIKE '%Count%';
```

The output of the above statement is shown in the widget below:

Output of SHOW FUNCTION STATUS

```
mysql> SHOW FUNCTION STATUS
-> LIKE '%Count%';
```

Db	Name	Type	Definer	Modified	Comment
MovieIndustry	DigitalAssetCount	FUNCTION	root@localhost	2020-06-02 06:22:09	
sys	ps_is_account_enabled	FUNCTION	mysql.sys@localhost	2020-06-02 04:51:44	

Determines whether instrumentation of an account is enabled within Performance Schema.

Parameters

in_host VARCHAR(60):
The hostname of the account to check.
in_user VARCHAR(32):
The username of the account to check.

Returns

ENUM('YES', 'NO', 'PARTIAL')

Example

```
mysql> SELECT sys.ps_is_account_enabled('localhost', 'root');
```

sys.ps_is_account_enabled('localhost', 'root')
YES

1 row in set (0.01 sec)

sys	ps thread account	FUNCTION	mysql.sys@localhost	2020-06-02 04:51:44	20
-----	-------------------	----------	---------------------	---------------------	----

Description

Return the user@host account for the given Performance Schema thread id.

Parameters

in_thread_id (BIGINT UNSIGNED):

The id of the thread to return the account for.

Example

```
mysql> select thread_id, processlist_user, processlist_host from performance_schema.threads where
```

thread_id	processlist_user	processlist_host
23	NULL	NULL
30	root	localhost
31	msandbox	localhost
32	msandbox	localhost

4 rows in set (0.00 sec)

```
mysql> select sys.ps_thread_account(31);
```

sys.ps_thread_account(31)
msandbox@localhost

1 row in set (0.00 sec)

utf8	utf8_general_ci	utf8_general_ci
------	-----------------	-----------------

3 rows in set (0.01 sec)

3. The function DigitalAssetCount can be called in any SQL statement.

```
SELECT Id, DigitalAssetCount(Id) AS Count
FROM Actors;
```

4. The function can also be called from stored procedures. The code below creates a stored procedure **GetDigitalAssetCount** that calls the **DigitalAssetCount** function.

```
DELIMITER **

CREATE PROCEDURE GetDigitalAssetCount(
    IN ActorNo INT,
    OUT Message VARCHAR(50))
BEGIN
```

```

DECLARE Number INT DEFAULT 0;

SET Number = ActorNo;

SET Message = DigitalAssetCount(Number);
END**

DELIMITER ;

```

This is a contrived example where the stored procedure is called with an actor ID. This ID is then used inside the stored procedure to call the stored function.

```

CALL GetDigitalAssetCount(10,@assetcount);
SELECT @assetcount;

```

5. The **DigitalAssetCount** function is a deterministic function as it returns the same output when the function is called with the same input parameter. Nondeterministic functions are those that may return different outputs for the same input. This is because they use **NOW()**, **RAND()**, or any other similar function. As an example, we will create a function **TimeSinceLastUpdate** to find how much time has elapsed since an actor updated a particular digital asset. The function takes two input parameters, Actor ID and the Asset type and returns an **INT** value as the number of seconds that have passed.

```

DELIMITER **

CREATE FUNCTION TimeSinceLastUpdate(
    ID INT,
    Asset VARCHAR(15))
RETURNS INT
NOT DETERMINISTIC
BEGIN
    DECLARE ElapsedTime INT;

    SELECT TIMESTAMPDIFF(SECOND, LastUpdatedOn, NOW())
    INTO ElapsedTime
    FROM DigitalAssets
    WHERE ActorID = ID AND AssetType = Asset;

    RETURN ElapsedTime;
END**

```

```
END**  
DELIMITER ;
```

In the function body, we declare a variable for the elapsed time value and then use the **TIMESTAMPDIFF** function. We have chosen **SECOND** as the unit for **TIMESTAMPDIFF** as it clearly demonstrates the non deterministic nature of our function.

```
SELECT TimeSinceLastUpdate(1, 'Instagram');
```

Calling this function again and again will result in a different output.

6. As mentioned in the lesson on creating and listing stored procedures, the **ROUTINES** tables in the `information_schema` database contains information about all functions in all the databases. The following query shows all the functions in the **classicmodels** database:

```
SELECT routine_name  
FROM information_schema.routines  
WHERE routine_type = 'FUNCTION' AND routine_schema = 'MovieIndustry';
```

7. We will now see how to remove this stored function.

```
DROP FUNCTION DigitalAssetCount;  
DROP FUNCTION IF EXISTS DigitalAssetCount;
```

When we try to drop a function that does not exist, a warning is issued. We can view the warning details using the **SHOW WARNINGS** statement:

```
SHOW WARNINGS;
```