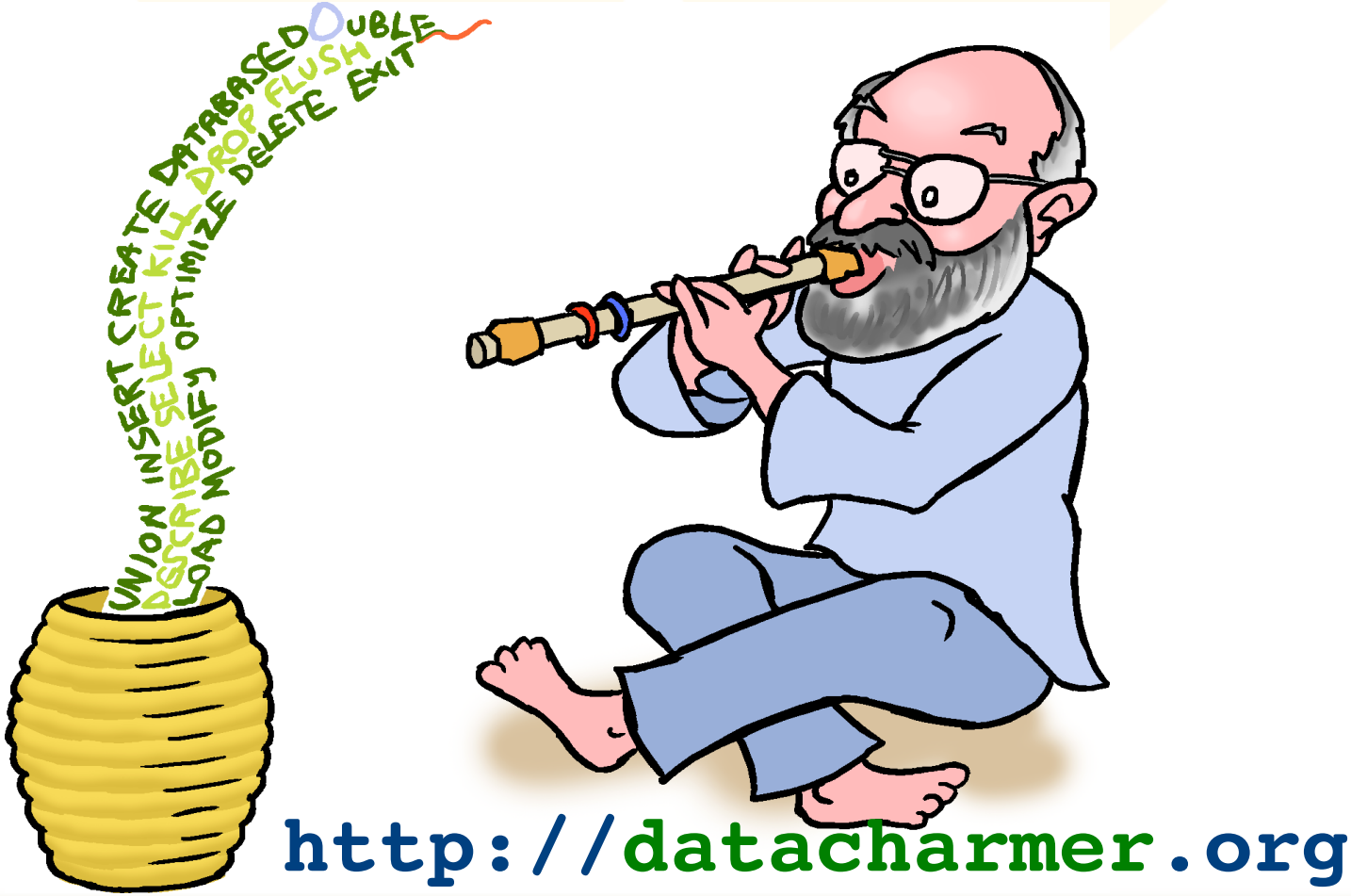
The MySQL General Purpose Routine library

Giuseppe Maxia

OPEN SOURCE DATABASE
CONFERENCE 2006
Learn • Share • Network • Benefit

About me



http://datacharmer.org

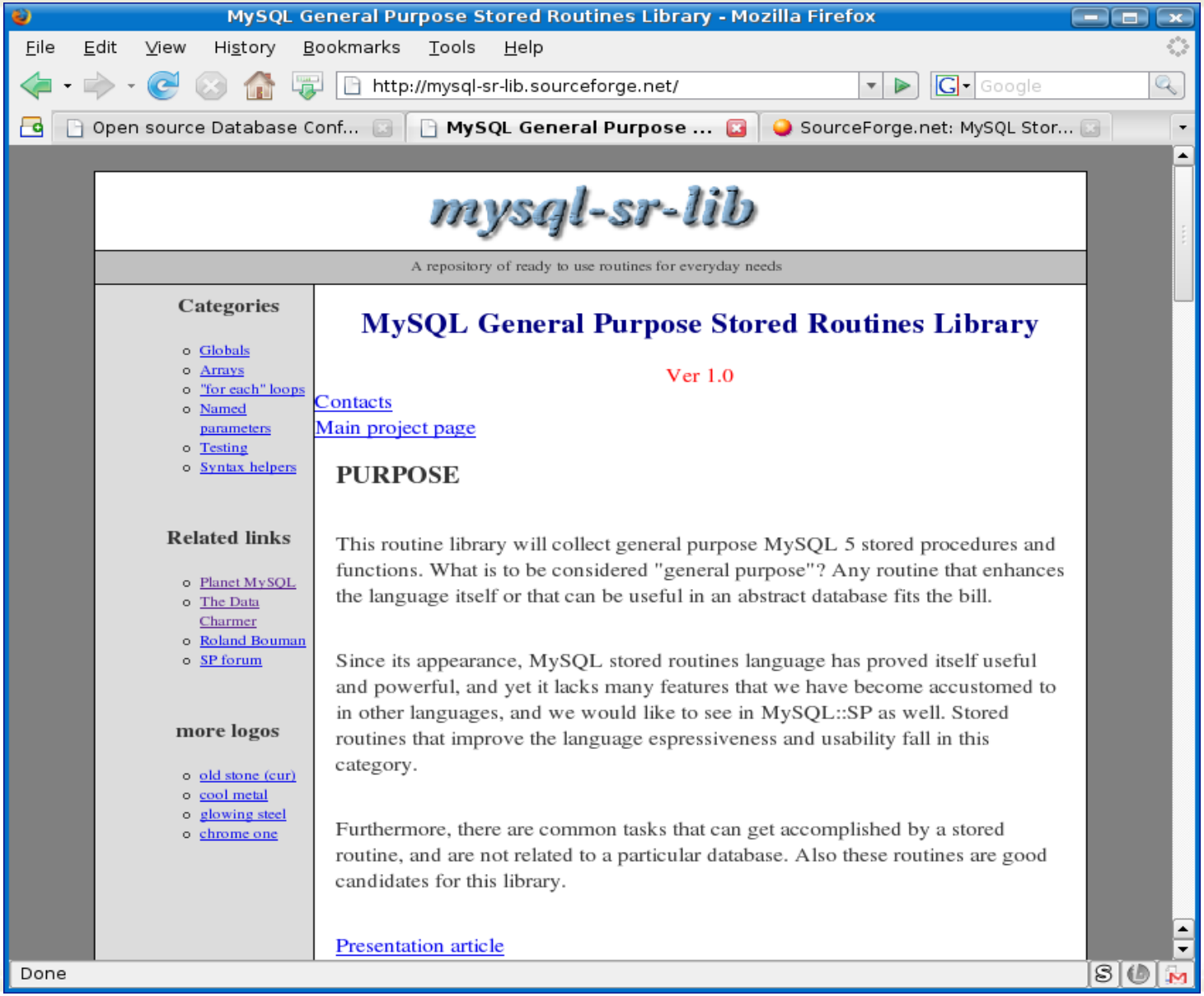
MySQL General Purpose Stored Routines Library

- A public repository
- under GPL license
- https://sf.net/projects/mysql-sr-lib/
- routines for general database usage
- IT IS NOT FROM MySQL AB!

Components

- global variables
- simple and complex data structures
- loops
- syntax helpers
- named parameters
- test units

- `globals`
- `arrays`
- `for_each_loops`
- `syntax`
- `simple_sp`
- `test_utilities`


MySQL General Purpose Stored Routines Library - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://mysql-sr-lib.sourceforge.net/
Google
Open source Database Conf...
MySQL General Purpose ...
SourceForge.net: MySQL Stor...
mysql-sr-lib
A repository of ready to use routines for everyday needs
Categories
Globals
Arrays
"for each" loops
Named parameters
Testing
Syntax helpers
Related links
Planet MySQL
The Data Charmer
Roland Bouman
SP forum
more logos
old stone (cur)
cool metal
glowing steel
chrome one
MySQL General Purpose Stored Routines Library
Ver 1.0
Contacts
Main project page
PURPOSE
This routine library will collect general purpose MySQL 5 stored procedures and functions. What is to be considered "general purpose"? Any routine that enhances the language itself or that can be useful in an abstract database fits the bill.
Since its appearance, MySQL stored routines language has proved itself useful and powerful, and yet it lacks many features that we have become accustomed to in other languages, and we would like to see in MySQL::SP as well. Stored routines that improve the language espressiveness and usability fall in this category.
Furthermore, there are common tasks that can get accomplished by a stored routine, and are not related to a particular database. Also these routines are good candidates for this library.
Presentation article
Done

INSTALLATION

- create a database
- download the library from https://sf.net/projects/mysql-sr-lib/
- run library.mysql
- run the tests under the ./test directory

INSTALLATION (1)

```
$ mysql -e 'create schema lib'
$ mysql lib < library.mysql
globals
test_utilities
arrays
for each
simple sp
syntax
```

INSTALLATION (2)

```
$ cd tests
$ mysql -t lib < test_library.mysql
...
+-------------+--------+--------+-------------------------+
| total tests | passed | failed | passed tests percentage |
+-------------+--------+--------+-------------------------+
|          56 |     56 |      0 | 100.00                  |
+-------------+--------+--------+-------------------------+
...
+-------------+--------+--------+-------------------------+
| total tests | passed | failed | passed tests percentage |
+-------------+--------+--------+-------------------------+
|           8 |      8 |      0 | 100.00                  |
+-------------+--------+--------+-------------------------+
```

Docs

- reference manual on the library web site
- Articles on http://datacharmer.blogspot.com
- Embedded syntax (we'll see it later)

globals (1)

- persistent variables (aren't invalidated when the session ends)
- Multi-user variables (each user can name his/her variables without conflicts)
- general utility functions

globals (2)

```
call global_var_set('max_val',10);
set @max_val=10;

select global_var_get('max_val'),
@max_val;
+-----------------------------+-----------+
| global_var_get('max_val') | @max_val  |
+-----------------------------+-----------+
| 10                          | 10        |
+-----------------------------+-----------+
```

globals (3)

```
connect;
Connection id:    416
Current database: lib
select global_var_get('max_val'),
@max_val;
+-----------------------------+-----------+
| global_var_get('max_val') | @max_val  |
+-----------------------------+-----------+
| 10                          | NULL      |
+-----------------------------+-----------+
```

globals (4)

```
drop table if exists t1;
Query OK, 0 rows affected, 1 warning
(0.01 sec)

select table_exists(schema(), 't1');
+-------------------------------+
| table_exists(schema(), 't1') |
+-------------------------------+
|                             0 |
+-------------------------------+
```

globals (5)

```
create table t1 (i int);
Query OK, 0 rows affected (0.01 sec)

select table_exists(schema(), 't1');
+-----------------------------+
| table_exists(schema(), 't1') |
+-----------------------------+
|                           1 |
+-----------------------------+
```

globals (6)

```
# more functions

view_exists(db, nome)
table_or_view_exists(db, nome)
function_exists(db,nome)
procedure exists(db,nome
routine_exists(db,nome,tipo)

global_var_drop(nome)
global_var_exists(nome)
```

arrays (1)

- persistent vectors
- Multi-user structures (each user can name his/her variables without conflicts)
- each array is at the same time a list, a hash, a queque and a heap
- sorting and merging routines

arrays (2)

```
call array_set('myvals', 1, '100');
call array_set('myvals', 'foo', 200);

call array_show('myvals');
+--------------+-----------+------------------+
| array_index | array_key | "myvals" values |
+--------------+-----------+------------------+
|            1 | NULL      | 100              |
|            2 | foo       | 200              |
+--------------+-----------+------------------+
```

arrays (3)

```
select array_get('myvals',2);
+-------------------------+
| array_get('myvals',2) |
+-------------------------+
| 200                     |
+-------------------------+
select array_get('myvals','foo');
+-----------------------------+
| array_get('myvals','foo') |
+-----------------------------+
| 200                         |
+-----------------------------+
```

arrays (as heaps) (4)

```
call array_drop('myvals');
call array_push('myvals', '100');
call array_push('myvals', 200);

call array_show('myvals');
```

```
+--------------+-----------+------------------+
| array_index | array_key | "myvals" values |
+--------------+-----------+------------------+
|            0 | NULL      | 100              |
|            1 | NULL      | 200              |
+--------------+-----------+------------------+
```

arrays (as heaps) (5)

```
select array_pop('myvals');
+----------------------+
| array_pop('myvals') |
+----------------------+
| 200                  |
+----------------------+
select array_pop('myvals');
+----------------------+
| array_pop('myvals') |
+----------------------+
| 100                  |
+----------------------+
```

arrays (as heaps) (6)

```
select array_pop('myvals');
+----------------------+
| array_pop('myvals') |
+----------------------+
| NULL                 |
+----------------------+
call array_show('myvals');
empty set (0.0 sec)
```

arrays (from list) (7)

```
call
array_from_list('100,200,300','myvals');

call array_show('myvals');
+--------------+-----------+------------------+
| array_index | array_key | "myvals" values |
+--------------+-----------+------------------+
|            0 | NULL      | 100              |
|            1 | NULL      | 200              |
|            2 | NULL      | 300              |
+--------------+-----------+------------------+
```

arrays (from list) (8)

```
call array_from_pair_list(
'foo=>100,bar=>200,baz=>300','myvals');

call array_show('myvals');
+--------------+-----------+-----------------+
| array_index | array_key | "myvals" values |
+--------------+-----------+-----------------+
|            0 | foo       | 100             |
|            1 | bar       | 200             |
|            2 | baz       | 300             |
+--------------+-----------+-----------------+
```

arrays (9)

```
# more routines
array_exists(name)
array_create(name, how_many)
array_clear(nome)
array_get_by_index(name,num)
array_get_by_key(name,key)
array_set_by_index(name,num)
array_set_by_key(name,key)

# more : copy, merge, append, sort,
grep, etc
```

„for_each“ loops (1)

- on-the-fly loops
- predefined loops
  - counters
  - set of tables
  - table rows
  - array elements

„for_each“ loops (2)

```
call for_each_counter(1,3,1,'create
table foo$N(i int)');

show tables like 'foo%';
+-----------------------+
| Tables_in_lib (foo%) |
+-----------------------+
| foo1                  |
| foo2                  |
| foo3                  |
+-----------------------+
```

„for_each“ loops (3)

```
call for_each_table(schema(),
'table_name like "foo%"',
'insert into $D.$T values ($N)');
```

```
select * from foo1;
+------+
| i    |
+------+
|    1 |
+------+
```

```
select * from foo2;
+------+
| i    |
+------+
|    2 |
+------+
```

„for_each“ loops (4)

```
call for_each_counter(1,3,1,'drop table
foo$N');

show tables like 'foo%';
empty set (0.00 sec)
```

syntax (1)

- provide usage syntax for each routine
- list of routines with filters
- support for named parameters (next section)

syntax (2)

```
call my_routines('global');
```

```
+----------------+--------------------+--------------+
| routine_schema | routine_name       | routine_type |
+----------------+--------------------+--------------+
| lib            | global_var_drop    | function     |
| lib            | global_var_drop    | procedure    |
| lib            | global_var_exists  | function     |
| lib            | global_var_get     | function     |
| lib            | global_var_set     | function     |
| lib            | global_var_set     | procedure    |
+----------------+--------------------+--------------+
```

syntax (3)

```
call my_procedures('global');
```

```
+----------------+--------------------+
| routine_schema | routine_name       |
+----------------+--------------------+
| lib            | global_var_drop    |
| lib            | global_var_set     |
+----------------+--------------------+
```

syntax (4)

```
call my_functions('global');
```

```
+----------------+--------------------+
| routine_schema | routine_name       |
+----------------+--------------------+
| lib            | global_var_drop    |
| lib            | global_var_exists  |
| lib            | global_var_get     |
| lib            | global_var_set     |
+----------------+--------------------+
```

syntax (5)

```
select fsyntax('global_var_get')\G
 ** function lib.global_var_get

    function global_var_get
        p_var_name varchar(50)
    --
    Returns the value of a global variable p_var_name
    --
    returns the variable value, null if variable does
not exist
   --
   see also:  global_var_set, global_var_exists
   --
   module : globals
```

simple_sp (1)

- lets you call routines with named parameters, rather than „by position“
- requires the „syntax“ module to be installed as well

simple_sp (2)

```
select psyntax('for_each_counter_complete')\G

    procedure for_each_counter_complete
        counter_start   INT,              # 1
        counter_end     INT,              # 2
        counter_delta   INT,              # 3
        sql_command     text,             # 4
        sql_before      text,             # 5
        sql_after       text,             # 6
        ba_mode enum("once","many")  # 7

# 7 parameters!
```

simple_sp (3)

```
# calling parameters by position
call for_each_counter_complete(
   1,
   10,
   1,
   'set @mycnt = @mycnt + 1',
   'set @mycnt = 0',
   'set @mycnt := @mycnt * @mycnt',
   'once');
```

simple_sp (3)

```
# calling parameters by name
call simple_spl(
     'for_each_counter_complete',
     'sql_before=>set @mycnt = 0;
      counter_start=>1;
      counter_end=>10;
      counter_delta=>1;
      sql_command=>set @mycnt = @mycnt + 1;
      sql_after=>set @mycnt := @mycnt * @mycnt;
      ba_mode=>once'
);
```

test_utilities (1)

- implements utility routines to create unit tests
- good programming practice

test_utilities (2)

```
# test_tutorial.mysql (1)

call initialize_tests();

call check_routine(
  database(), 'make_table', 'procedure');

call check_routine(
  database(), 'get_result', 'function');
```

test_utilities (3)

```
# test_tutorial.mysql (2)
connect;
insert into my_data values
  ('first', 1, 2),
  ('second',10,20),
  ('third', 100, 200);

set @my_result = get_result('first');

call log_test('get_result first',
  @my_result,
  '@my_result = 3',
  @my_result = 3 );
```

test_utilities (4)

```
# test_tutorial.mysql (3)

set @my_result = get_result('second');

call log_test('get_result second',
  @my_result,
  '@my_result = 30', @my_result = 30 );

set @my_result = get_result('third');

call log_test('get_result third',
  @my_result,
  '@my_result = 300', @my_result = 300 );
```

test_utilities (5)

```
# test_tutorial.mysql (4)
set @my_result = get_result('fourth');

call log_test('get_result non-existent',
  @my_result,
  '@my_result is null',
  @my_result is null );

call show_test_results();

# So far, the routine to test is yet
# to be created!
```

test_utilities (6)

```
# test1.mysql (1)
delimiter $$

drop procedure if exists make_table $$
create procedure make_table ()
deterministic
begin
    drop table if exists my_data;
    create table my_data
    (   name char(10) not null primary key,
        val1   int,
        val2 int );
end $$
```

test_utilities (7)

```
# test1.mysql (2)
drop function if exists get_result $$
create function get_result ( p_name char(10))
returns int
reads sql data
begin
    declare mysum int;
    select val1 * val2 into mysum # mistake !
    from my_data
    where name = p_name;
    return mysum;
end $$

delimiter ;
```

test_utilities (8)

```
# Let's try it! (1)
$ mysql lib < examples/test1.mysql
$ mysql -t lib < examples/test_tutorial.mysql
```

test no	description	result	expected	outcome
1	make_table p exists	1	routine_exists = true	pass
2	get_result f exists	1	routine_exists = true	pass
3	table my_data exists	1	table_exists = true	pass
4	get_result first	2	@my_result = 3	** fail **
5	get_result second	200	@my_result = 30	** fail **
6	get_result third	20000	@my_result = 300	** fail **
7	get_result non-existent	NULL	@my_result is null	pass

test_utilities (9)

```
# Let's try it! (2)
$ mysql -t lib < examples/test_tutorial.mysql
...
+-------------+--------+--------+-------------------------+
| total tests | passed | failed | passed tests percentage |
+-------------+--------+--------+-------------------------+
|           7 |      4 |      3 | 57.14                   |
+-------------+--------+--------+-------------------------+
+---------+-------------------+--------+------------------+------------+
| test no | description       | result | expected         | outcome    |
+---------+-------------------+--------+------------------+------------+
|       4 | get_result first  | 2      | @my_result = 3   | ** fail ** |
|       5 | get_result second | 200    | @my_result = 30  | ** fail ** |
|       6 | get_result third  | 20000  | @my_result = 300 | ** fail ** |
+---------+-------------------+--------+------------------+------------+
```

test_utilities (7)

```
# let's fix the mistake
drop function if exists get_result $$
create function get_result ( p_name char(10))
returns int
reads sql data
begin
    declare mysum int;
    select val1 + val2 into mysum # correct !
    from my_data
    where name = p_name;
    return mysum;
end $$

delimiter ;
```

test_utilities (8)

```
# Let's try again
$ mysql lib < examples/test1.mysql
$ mysql -t lib < examples/test_tutorial.mysql
+---------+-------------------------+--------+----------------------+---------+
| test no | description             | result | expected             | outcome |
+---------+-------------------------+--------+----------------------+---------+
|       1 | make_table p exists     | 1      | routine_exists = true | pass    |
|       2 | get_result f exists     | 1      | routine_exists = true | pass    |
|       3 | table my_data exists    | 1      | table_exists = true   | pass    |
|       4 | get_result first        | 3      | @my_result = 3        | pass    |
|       5 | get_result second       | 30     | @my_result = 30       | pass    |
|       6 | get_result third        | 300    | @my_result = 300      | pass    |
|       7 | get_result non-existent | NULL   | @my_result is null    | pass    |
+---------+-------------------------+--------+----------------------+---------+
+-------------+--------+--------+-------------------------+
| total tests | passed | failed | passed tests percentage |
+-------------+--------+--------+-------------------------+
|           7 |      7 |      0 | 100.00                  |
+-------------+--------+--------+-------------------------+
```

THANKS

Any questions?

http://datacharmer.org