Inexpensive Datamasking for MySQL with ProxySQL

data anonymization for developers

FOSDEM MySQL & Friends Devroom - February 2017

René Cannaò - ProxySQL Founder

Frédéric Descamps - MySQL Community Manager - Oracle

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purpose only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied up in making purchasing decisions. The development, release and timing of any features or functionality described for Oracle's product remains at the sole discretion of Oracle.

Who are we ?

René Cannaò

- @rene_cannao
- ProxySQL Founder

Frédéric Descamps

- @lefred
- MySQL Evangelist
- Managing MySQL since 3.23
- devops believer

What is ProxySQL ?

What is ProxySQL ?

the MySQL data stargate

Why using ProxySQL as datamasking solution?

- Open Source & Free like in beer

Why using ProxySQL as datamasking solution?

- Open Source & Free like in beer
- Other solutions are expensive or not working

Why using ProxySQL as datamasking solution?

- Open Source & Free like in beer
- Other solutions are expensive or not working
- Not worse than the other solutions as currently none is perfect

Why using ProxySQL as datamasking solution?

- Open Source & Free like in beer
- Other solutions are expensive or not working
- Not worse than the other solutions as currently none is perfect
    - the best solution would be to have this feature implemented in the server just after the handler API

The concept

We use Regular Expressions to modify the client's SQL statement and replace the column(s) we want to hide by some characters.

Only the defined users, in our example, we use a developer will have his statements modified.

Access

don't forget to create a user.

```
> insert into mysql_users
  (username, password, active, default_hostgroup)
  values ('devel','devel',1,1);
```

Rules

Avoid `SELECT *`

Rules

Avoid `SELECT *`

- we need to create some rules to block any `SELECT *` variant on the table

Rules

Avoid `SELECT *`

- we need to create some rules to block any `SELECT *` variant on the table
- if the column is part of many tables, we need to do so for each of them

Rules (2)

Mask the field

Rules (2)

Mask the field

when the field is selected in the columns we need:

Rules (2)

Mask the field

when the field is selected in the columns we need:

- to replace the columnn by showing the first 2 characters and a certain amount of Xs

Rules (2)

Mask the field

when the field is selected in the columns we need:

- to replace the columnn by showing the first 2 characters and a certain amount of Xs
- keep the column name

Rules (2)

Mask the field

when the field is selected in the columns we need:

- to replace the columnn by showing the first 2 characters and a certain amount of Xs
- keep the column name

```
5275653223285289 will become 52XXXXXXXXX
```

Rules Overview

Too mask `cc_num` from table `CUSTOMERS`, 7 rules are needed:

Rules Overview

Too mask `cc_num` from table `CUSTOMERS`, 7 rules are needed:

rule #1

```
          rule_id: 1
           active: 1
         username: devel
           flagIN: 0
    match_pattern: `*cc_num*`
    re_modifiers: caseless,global
          flagOUT: NULL
  replace_pattern: cc_num
            apply: 0
```

rule #2

```
         rule_id: 2
          active: 1
        username: devel
          flagIN: 0
   match_pattern: (\(?)(`?\w+`?\.)?cc_num(\)?)([ ,\n])
   re_modifiers: caseless,global
         flagOUT: NULL
replace_pattern:
     \1CONCAT(LEFT(\2cc_num,2),REPEAT('X',10))\3 cc_num\4
           apply: 0
```

rule #3

```
          rule_id: 3
           active: 1
         username: devel
           flagIN: 0
    match_pattern: \)(\)?) cc_num\s+(\w),
    re_modifiers: caseless,global
          flagOUT: NULL
  replace_pattern: )\1 \2,
            apply: 1
```

rule #4

```
         rule_id: 4
          active: 1
        username: devel
          flagIN: 0
   match_pattern: \)(\)?) cc_num\s+(.*)\s+from
   re_modifiers: caseless,global
         flagOUT: NULL
replace_pattern: )\1 \2 from
           apply: 1
```

rule #5

```
       rule_id: 5
        active: 1
      username: devel
match_pattern: ^SELECT\s+\*.*FROM.*CUSTOMERS
 re_modifiers: caseless,global
     error_msg: Query not allowed due to sensitive
                information, please contact dba@myapp.com
         apply: 0
```

rule #6

```
      rule_id: 6
       active: 1
     username: devel
match_pattern: ^SELECT\s+CUSTOMERS\.\*.*FROM.*CUSTOMERS
 re_modifiers: caseless,global
    error_msg: Query not allowed due to sensitive
               information, please contact dba@myapp.com
        apply: 0
```

rule #7

```
      rule_id: 7
       active: 1
     username: devel
match_pattern:
      ^SELECT\s+(\w+)\.\*.*FROM.*CUSTOMERS\s+(as\s+)?(\1)
 re_modifiers: caseless,global
    error_msg: Query not allowed due to sensitive
               information, please contact dba@myapp.com
        apply: 0
```

Limitations

- supported in proxySQL >= 1.4.x

Limitations

- supported in proxySQL >= 1.4.x
- all fields with the same name will be masked whatever the name of the table is

Limitations

- supported in proxySQL >= 1.4.x
- all fields with the same name will be masked whatever the name of the table is
- the regexps can always be not sufficient

Make it easy

This is not really easy isn't it ?

You can use this small bash script (https://gist.github.com/lefred/c040fee7e9c60ff3ca80f1590c48572b) to generate them:

```
# ./maskit.sh -c cc_num -t CUSTOMERS
column: cc_num
table: CUSTOMERS
let's add the rules...
```

Examples

Easy ones

- `SELECT * FROM CUSTOMERS;`
- `SELECT firstname, lastname, cc_num FROM CUSTOMERS;`

Examples (2)

More difficult

Thank you Thomas Adolph & Dipti Joshi for the suggestions

```
select firstname, CONCAT(cc_num), lastname from
myapp.CUSTOMERS;

select firstname, cc_num, cc_num from myapp.CUSTOMERS;

select firstname, `cc_num` from myapp.CUSTOMERS;

select firstname, cc_num
from myapp.CUSTOMERS; (*)
```

(*) on two lines

Examples (3)

```
select t1.cc_num from myapp.CUSTOMERS as t1;

select firstname, cc_num as fred from CUSTOMERS;

select firstname, cc_num fred from CUSTOMERS;

select firstname, cc_num `as` from CUSTOMERS;
select cc_num as `as`, firstname from CUSTOMERS;
select `t1`.`cc_num` from myapp.CUSTOMERS as t1;
```

Examples (4)

```
select cc_num fred, firstname from CUSTOMERS;

select firstname, /* cc_num */, from myapp.CUSTOMERS;

/* */ select firstname, cc_num from myapp.CUSTOMERS;

select CUSTOMERS.* from myapp.CUSTOMERS;

select a.* from myapp.CUSTOMERS a;
```

We need you !

Thank you !

Questions ?