MySQL Sharding with ProxySQL

30 Aug 2016

By Marco Tusa | MySQL | MySQL, proxy, ProxySQL, sharding | 8 Comments

This article demonstrates how MySQL sharding with ProxySQL works.

Recently a colleague of mine asked me to provide a simple example on how ProxySQL performs sharding.

In response, I'm writing this short tutorial in the hope it will illustrate ProxySQL's sharding functionalities, and help people out there better understand how to use it.

ProxySQL is a very powerful platform that allows us to manipulate and manage our connections and queries in a simple but effective way. This article shows you how.

Before starting let's clarify some basic concepts.

- ProxySQL organizes its internal set of servers in Host Groups (HG), and each HG can be associated with users and Query Rules (QR)
- Each QR can be final (apply = 1) or = let ProxySQL continue to parse other QRs
- A QR can be a rewrite action, be a simple match, have a specific target HG, or be generic
- QRs are defined using regex

You can see QRs as a sequence of filters and transformations that you can arrange as you like.

These simple basic rules give us enormous flexibility. They allow us to create very simple actions like a simple query re-write, or very complex chains that could see dozens of QR concatenated. Documentation can be found here.

The information related to HGs or QRs is easily accessible using the ProxySQL administrator interface, in the tables mysql_servers, mysql_query_rules and stats.stats_mysql_query_rules. The last one allows us to evaluate if and how the rule(s) is used.

With regards to sharding, what can ProxySQL do to help us achieve what we need (in a relatively easy way)? Some people/companies include sharding logic in the application, use multiple connections to reach the different targets, or have some logic to split the load across several schemas/tables. ProxySQL allows us to simplify the way connectivity and query distribution is supposed to work reading data in the query or accepting HINTS.

No matter what the requirements, the sharding exercise can be summarized in a few different categories.

- By splitting the data inside the same container (like having a shard by State where each State is a schema)
- By physical data location (this can have multiple MySQL servers in the same room, as well as having them geographically distributed)
- A combination of the two, where I do split by State using a dedicated server, and again split by schema/table by whatever (say by gender)

In the following examples, I show how to use ProxySQL to cover the three different scenarios defined above (and a bit more).

The example below will report text from the Admin ProxySQL interface and the MySQL console. I will mark each one as follows:

- Mc for MySQL console
- Pa for ProxySQL Admin

Please note that the MySQL console MUST use the **-c** flag to pass the comments in the query. This is because the default behavior in the MySQL console is to remove the comments.

For the example described below I have a PrxySQL v1.2.2 that is going to become the master in few days. You can download it from:

```Shell
git clone https://github.com/sysown/proxysql.git
git checkout v1.2.2
```

Then to compile:

```Shell
cd <path to proxy source code>
make
make install
```

If you need full instructions on how to install and configure ProxySQL, read here and here.

Finally, you need to have the WORLD test DB loaded. WORLD test DB can be found here.

Shard inside the same MySQL Server using three different schemas split by continent

Obviously, you can have any number of shards and relative schemas. What is relevant here is demonstrating how traffic gets redirected to different targets (schemas), maintaining the same structure (tables), by discriminating the target based on some relevant information in the data or pass by the application.

OK, let us roll the ball.

```MySQL
[Mc]
+---------------+-------------+
| Continent     | count(Code) |
+---------------+-------------+
| Asia          |          51 | <--
| Europe        |          46 | <--
| North America |          37 |
| Africa        |          58 | <--
| Oceania       |          28 |
| Antarctica    |           5 |
| South America |          14 |
+---------------+-------------+
```

For this exercise, I will use three hosts in replica.

To summarize, I will need:

- Three hosts: 192.168.1.[5-6-7]
- Three schemas: Continent X + world schema
- One user : user_shardRW
- Three hostgroups: 10, 20, 30 (for future use)

First, we will create the schemas Asia, Africa, Europe:

```Shell
[Mc]
Create schema [Asia|Europe|North_America|Africa];
create table Asia.City as select a.* from  world.City a join Country on a.CountryCode = Country.code where Continent='Asi
create table Europe.City as select a.* from  world.City a join Country on a.CountryCode = Country.code where Continent='E
create table Africa.City as select a.* from  world.City a join Country on a.CountryCode = Country.code where Continent='A
create table North_America.City as select a.* from  world.City a join Country on a.CountryCode = Country.code where Conti

create table Asia.Country as select * from  world.Country where Continent='Asia' ;
create table Europe.Country as select * from  world.Country where Continent='Europe' ;
create table Africa.Country as select * from  world.Country  where Continent='Africa' ;
create table North_America.Country as select * from  world.Country where Continent='North America' ;
```

Now, create the user

```MySQL
grant all on *.* to user_shardRW@'%' identified by 'test';
```

Now let us start to configure the ProxySQL:

```MySQL
```

```
4
5  INSERT INTO mysql_servers (hostname,hostgroup_id,port,weight) VALUES ('192.168.1.5',10,3306,100);
6  INSERT INTO mysql_servers (hostname,hostgroup_id,port,weight) VALUES ('192.168.1.6',20,3306,100);
7  INSERT INTO mysql_servers (hostname,hostgroup_id,port,weight) VALUES ('192.168.1.7',30,3306,100);
8  LOAD MYSQL SERVERS TO RUNTIME; SAVE MYSQL SERVERS TO DISK;
```

With this we have defined the user, the servers and the host groups.

Let us start to define the logic with the query rules:

MySQL

```
1  [Pa]
2  delete from mysql_query_rules where rule_id > 30;
3  INSERT INTO mysql_query_rules (rule_id,active,username,match_pattern,replace_pattern,apply) VALUES (31,1,'user_shardRW',"^
4  LOAD MYSQL QUERY RULES TO RUNTIME;SAVE MYSQL QUERY RULES TO DISK;
```

I am now going to query the master (or a single node), but I am expecting ProxySQL to redirect the query to the right shard, catching the value of the continent:

MySQL

```
1  [Mc]
2   SELECT name,population from world.City  WHERE Continent='Europe' and CountryCode='ITA' order by population desc limit 1;
3  +------+------------+
4  | name | population |
5  +------+------------+
6  | Roma |    2643581 |
7  +------+------------+
```

You can say: “Hey! You are querying the schema World, of course you get back the correct data.”

This is not what really happened. ProxySQL did not query the schema World, but the schema Europe.

Let’s look at the details:

MySQL

```
1  [Pa]
2  select * from stats_mysql_query_digest;
3  Original    :SELECT name,population from world.City  WHERE Continent='Europe' and CountryCode='ITA' order by population de
4  Transformed :SELECT name,population from Europe.City WHERE ?=? and CountryCode=? order by population desc limit ?
```

Let me explain what happened.

Rule 31 in ProxySQL will take all the FIELDS we will pass in the query. It will catch the CONTINENT in the WHERE clause, it will take any condition after WHERE and it will reorganize the queries all using the RegEx.

Does this work for any table in the sharded schemas? Of course it does.

A query like: `SELECT name,population from world.Country WHERE Continent='Asia' ;`
Will be transformed into: `SELECT name,population from Asia.Country WHERE ?=?`

MySQL

```
1  [Mc]
2  +----------------------+------------+
3  | name                 | population |
4  +----------------------+------------+
5  | Afghanistan          |   22720000 |
6  | United Arab Emirates |    2441000 |
7  | Armenia              |    3520000 |
8  <snip ...>
9  | Vietnam              |   79832000 |
10 | Yemen                |   18112000 |
11 +----------------------+------------+
```

Another possible approach is to instruct ProxySQL to shard is to pass a hint inside a comment. Let see how.

First let me disable the rule I just inserted. This is not really needed, but we’ll do it so you can see how. 🙂

MySQL

```
1  [Pa]
2  mysql> update mysql_query_rules set active=0 where rule_id=31;
3  Query OK, 1 row affected (0.00 sec)
4  mysql> LOAD MYSQL QUERY RULES TO RUNTIME;SAVE MYSQL QUERY RULES TO DISK;
5  Query OK, 0 rows affected (0.00 sec)
```

Done.

Now what I want is for *ANY* query that contains the comment /* continent=X */ to go to the continent X schema, same server.

```
[Pa]
delete from mysql_query_rules where rule_id in (31,33,34,35,36);
INSERT INTO mysql_query_rules (rule_id,active,username,match_pattern,replace_pattern,apply,FlagOUT,FlagIN) VALUES (31,1,'u
INSERT INTO mysql_query_rules (rule_id,active,username,match_pattern,replace_pattern,apply,FlagIN,FlagOUT) VALUES (32,1,'u
INSERT INTO mysql_query_rules (rule_id,active,username,match_pattern,replace_pattern,apply,FlagOUT,FlagIN) VALUES (33,1,'u
INSERT INTO mysql_query_rules (rule_id,active,username,match_pattern,replace_pattern,apply,FlagIN,FlagOUT) VALUES (34,1,'u
INSERT INTO mysql_query_rules (rule_id,active,username,match_pattern,replace_pattern,apply,FlagOUT,FlagIN) VALUES (35,1,'u
INSERT INTO mysql_query_rules (rule_id,active,username,match_pattern,replace_pattern,apply,FlagIN,FlagOUT) VALUES (36,1,'u
LOAD MYSQL QUERY RULES TO RUNTIME;SAVE MYSQL QUERY RULES TO DISK;
```

How does this work?

I have defined two concatenated rules. The first captures the incoming query containing the desired value (like continent = Asia). If the match is there, ProxySQL exits that action, but while doing so it will read the Apply field. If Apply is 0, it will read the FlagOUT value. At this point it will go to the first rule (in sequence) that has the value of FlagIN equal to the FlagOUT.

The second rule gets the request and will replace the value of world with the one I have defined. In short, it replaces whatever is in the match_pattern with the value that is in the replace_pattern.

Now ProxySQL implements the Re2 Google library for RegEx. Re2 is very fast but has some limitations, like it does NOT support (at the time of the writing) the flag option **g**. In other words, if I have a select with many tables, and as such several "*world*", Re2 will replace ONLY the first instance.

As such, a query like:

```
Select /* continent=Europe */ * from world.Country join world.City on world.City.CountryCode=world.Country.Code where Coun
```

Will be transformed into:

```
Select /* continent=Europe */ * from Europe.Country join world.City on world.City.CountryCode=world.Country.Code where Cou
```

And fail.

The other day, Rene and I were discussing how to solve this given the lack of implementation in Re2. Finally, we opted for recursive actions.

What does this mean? It means that ProxySQL from v1.2.2 now has a new functionality that allows recursive calls to a Query Rule. The maximum number of iterations that ProxySQL can run is managed by the option (global variable) *mysql-query_processor_iterations*. *Mysql-query_processor_iterations* define how many operations a query process can execute as whole (from start to end).

This new implementation allows us to reference a Query Rule to itself to be executed multiple times.

If you go back you will notice that QR 34 has FlagIN and FlagOUT pointing to the same value of 25 and Apply =0. This brings ProxySQL to recursively call rule 34 until it changes ALL the values of the word **world**.

The result is the following:

```
[Mc]
Select /* continent=Europe */ Code, City.Name, City.population  from world.Country join world.City on world.City.CountryC
+------+---------------+------------+
| Code | Name          | population |
+------+---------------+------------+
| RUS  | Moscow        |    8389200 |
| GBR  | London        |    7285000 |
| RUS  | St Petersburg |    4694000 |
| DEU  | Berlin        |    3386667 |
| ESP  | Madrid        |    2879052 |
+------+---------------+------------+
```

You can see ProxySQL internal information using the following queries:

```
[Pa]
 select active,hits, mysql_query_rules.rule_id, match_digest, match_pattern, replace_pattern, cache_ttl, apply,flagIn,fla
+--------+------+---------+--------------------+----------------------------------------+-----------------+-----------+-
| active | hits | rule_id | match_digest       | match_pattern                          | replace_pattern | cache_ttl |
+--------+------+---------+--------------------+----------------------------------------+-----------------+-----------+-
| 1      | 1    | 33      | NULL               | \S*\s*\/\*\s*continent=.*Europe\s*\*.* | NULL            | NULL      |
| 1      | 4    | 34      | NULL               | world.                                 | Europe.         | NULL      |
| 1      | 0    | 35      | NULL               | \S*\s*\/\*\s*continent=.*Africa\s*\*.* | NULL            | NULL      |
| 1      | 0    | 36      | NULL               | world.                                 | Africa.         | NULL      |
+--------+------+---------+--------------------+----------------------------------------+-----------------+-----------+-
```

MySQL

```
[Pa]
select * from stats_mysql_query_digest;
<snip and taking only digest_text>
Select Code, City.Name, City.population from Europe.Country join Europe.City on Europe.City.CountryCode=Europe.Country.Cod
```

As you can see ProxySQL has nicely replaced the word **world** in the query to Europe, and it ran Query Rule 34 four times (hits).

This is obviously working for Insert/Update/Delete as well.

Queries like:

MySQL

```
insert into  /* continent=Europe */  world.City values(999999,'AAAAAAA','ITA','ROMA',0) ;
```

Will be transformed into:

Shell

```
[Pa]
select digest_text from stats_mysql_query_digest;
+-------------------------------------------+
| digest_text                               |
+-------------------------------------------+
| insert into Europe.City values(?,?,?,?,?) |
+-------------------------------------------+
```

And executed only on the desired schema.

Sharding by host

Using hint

How can I shard and redirect the queries to a host (instead of a schema)? This is even easier! 🙂

The main point is that whatever matches the rule should go to a defined HG. No rewrite imply, which means less work.

So how this is done? As before, I have three NODES: 192.168.1.[5-6-7]. For this example, I will use world DB (no continent schema), distributed in each node, and I will retrieve the node bound to the IP to be sure I am going to the right place.

I instruct ProxySQL to send my query by using a HINT to a specific host. I choose the hint “shard_host_HG”, and I am going to inject it in the query as a comment.

As such the Query Rules will be:

MySQL

```
[Pa]
delete from mysql_query_rules where rule_id in (40,41,42, 10,11,12);
INSERT INTO mysql_query_rules (rule_id,active,username,match_pattern,destination_hostgroup,apply) VALUES (10,1,'user_shard
INSERT INTO mysql_query_rules (rule_id,active,username,match_pattern,destination_hostgroup,apply) VALUES (11,1,'user_shard
INSERT INTO mysql_query_rules (rule_id,active,username,match_pattern,destination_hostgroup,apply) VALUES (12,1,'user_shard
LOAD MYSQL QUERY RULES TO RUNTIME;SAVE MYSQL QUERY RULES TO DISK;
```

While the queries I am going to test are:

MySQL

```
[Mc]
Select /* shard_host_HG=Europe */ City.Name, City.Population from world.Country join world.City on world.City.CountryCode=
Select /* shard_host_HG=Asia */ City.Name, City.Population from world.Country join world.City on world.City.CountryCode=wo
Select /* shard_host_HG=Africa */ City.Name, City.Population from world.Country join world.City on world.City.CountryCode=
```

Running the query for Africa, I will get:

MySQL

```
| Name        | Population |
+-------------+------------+
| Addis Abeba |    2495000 |
| Dire Dawa   |     164851 |
| Nazret      |     127842 |
| Gonder      |     112249 |
| Dese        |      97314 |
+-------------+------------+
+---------------+----------------+
| VARIABLE_NAME | VARIABLE_VALUE |
+---------------+----------------+
| BIND_ADDRESS  | 192.168.1.7    |
+---------------+----------------+
```

That will give me:

```MySQL
[Pa]
select active,hits, mysql_query_rules.rule_id, match_digest, match_pattern, replace_pattern, cache_ttl, apply,flagIn,flagO
+--------+------+---------+--------------------+----------------------------------------+-----------------+-----------+--
| active | hits | rule_id | match_digest       | match_pattern                          | replace_pattern | cache_ttl | a
+--------+------+---------+--------------------+----------------------------------------+-----------------+-----------+--
| 1      | 0    | 40      | NULL               | \/\*\s*shard_host_HG=.*Europe\s*\*.    | NULL            | NULL      | 0
| 1      | 0    | 41      | NULL               | \/\*\s*shard_host_HG=.*Asia\s*\*.      | NULL            | NULL      | 0
| 1      | 2    | 42      | NULL               | \/\*\s*shard_host_HG=.*Africa\s*\*.    | NULL            | NULL      | 0
+--------+------+---------+--------------------+----------------------------------------+-----------------+-----------+--
```

In this example, we have NO replace_patter. This is only a matching and redirecting Rule, where the destination HG is defined in the value of the *destination_hostgroup* attribute while inserting. In the case for Africa, it is HG 30.

The server in HG 30 is:

```MySQL
[Pa]
select hostgroup_id,hostname,port,status from mysql_servers ;
+--------------+-------------+------+--------+
| hostgroup_id | hostname    | port | status |
+--------------+-------------+------+--------+
| 10           | 192.168.1.5 | 3306 | ONLINE |
| 20           | 192.168.1.6 | 3306 | ONLINE |
| 30           | 192.168.1.7 | 3306 | ONLINE | <---
+--------------+-------------+------+--------+
```

This perfectly matches our returned value.

You can try by your own the other two continents.

Using destination_hostgroup

Another way to assign a query's final host is to use the the destination_hostgroup, set the *Schema_name* attribute and use the *use schema* syntax in the query.

For example:

```MySQL
[Pa]
INSERT INTO mysql_query_rules (active,schemaname,destination_hostgroup,apply) VALUES
(1, 'shard00', 1, 1), (1, 'shard01', 1, 1), (1, 'shard03', 1, 1),
(1, 'shard04', 2, 1), (1, 'shard06', 2, 1), (1, 'shard06', 2, 1),
(1, 'shard07', 3, 1), (1, 'shard08', 3, 1), (1, 'shard09', 3, 1);
```

And then in the query do something like :

```MySQL
use shard02; Select * from tablex;
```

I mention this method because it is currently one of the most common in large companies using SHARDING.

But it is not safe, because it relays on the fact the query will be executed in the desired HG. The risk of error is high.

Just think if a query doing join against a specified SHARD:

```MySQL
use shard01; Select * from tablex join shard03 on tablex.id = shard03.tabley.id;
```

This will probably generate an error because shard03 is probably NOT present on the host containing shard01.

Shard by host and by schema

Finally, is obviously possible to combine the two approaches and shard by host with only a subset of schemas.

To do so, let's use all three nodes and have the schema distribution as follow:

- Europe on Server 192.168.1.5 -> HG 10
- Asia on Server 192.168.1.6 -> HG 20
- Africa on Server 192.168.1.7 -> HG 30

I have already set the query rules using HINT, so what I have to do is to use them BOTH to combine the operations:

MySQL

```
[Mc]
Select /* shard_host_HG=Asia */ /* continent=Asia */  City.Name, City.Population from world.Country join world.City on wo
+--------------------+------------+
| Name               | Population |
+--------------------+------------+
| Mumbai (Bombay)    |   10500000 |
| Delhi              |    7206704 |
| Calcutta [Kolkata] |    4399819 |
| Chennai (Madras)   |    3841396 |
| Hyderabad          |    2964638 |
+--------------------+------------+
5 rows in set (0.00 sec)

+---------------+----------------+
| VARIABLE_NAME | VARIABLE_VALUE |
+---------------+----------------+
| BIND_ADDRESS  | 192.168.1.6    |
+---------------+----------------+
1 row in set (0.01 sec)
```

MySQL

```
[Pa]
mysql> select digest_text from stats_mysql_query_digest;
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+
| digest_text                                                                                                                                                 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+
| SELECT * from information_schema.GLOBAL_VARIABLES where variable_name like ?                                                                                |
| Select City.Name, City.Population from Asia.Country join Asia.City on Asia.City.CountryCode=Asia.Country.Code where Cou                                    |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+
2 rows in set (0.00 sec)

mysql> select active,hits, mysql_query_rules.rule_id, match_digest, match_pattern, replace_pattern, cache_ttl, apply,flag
+--------+------+---------+--------------------+---------------------------------------+-----------------+-----------+-
| active | hits | rule_id | match_digest       | match_pattern                         | replace_pattern | cache_ttl |
+--------+------+---------+--------------------+---------------------------------------+-----------------+-----------+-
| 1      | 0    | 10      | NULL               | \/\*\s*shard_host_HG=.*Europe\s*\*.   | NULL            | NULL      |
| 1      | 2    | 11      | NULL               | \/\*\s*shard_host_HG=.*Asia\s*\*.     | NULL            | NULL      |
| 1      | 0    | 12      | NULL               | \/\*\s*shard_host_HG=.*Africa\s*\*.   | NULL            | NULL      |
| 1      | 0    | 13      | NULL               | NULL                                  | NULL            | NULL      |
| 1      | 1    | 31      | NULL               | \S*\s*\/\*\s*continent=.*Asia\s*\*.*  | NULL            | NULL      |
| 1      | 4    | 32      | NULL               | world.                                | Asia.           | NULL      |
| 1      | 0    | 33      | NULL               | \S*\s*\/\*\s*continent=.*Europe\s*\*.* | NULL            | NULL      |
| 1      | 0    | 34      | NULL               | world.                                | Europe.         | NULL      |
| 1      | 0    | 35      | NULL               | \S*\s*\/\*\s*continent=.*Africa\s*\*.* | NULL            | NULL      |
| 1      | 0    | 36      | NULL               | world.                                | Africa.         | NULL      |
+--------+------+---------+--------------------+---------------------------------------+-----------------+-----------+-
```

As you can see rule 11 has two HITS, which means my queries will go to the associated HG. But given that the Apply for rule 11 is =0, ProxySQL will first continue to process the Query Rules.

As such it will also transform the queries for rules 31 and 32, each one having the expected number of hits (one the first and the second four because of the loop).

That was all we had to do to perform a perfect two layer sharding in ProxySQL.

Conclusion

ProxySQL allows the user to access data distributed by shards in a very simple way. The query rules that follow the consolidate RegEx pattern, in conjunction with the possibility to concatenate rules and the Host Group approach definition give us huge flexibility with relative simplicity.