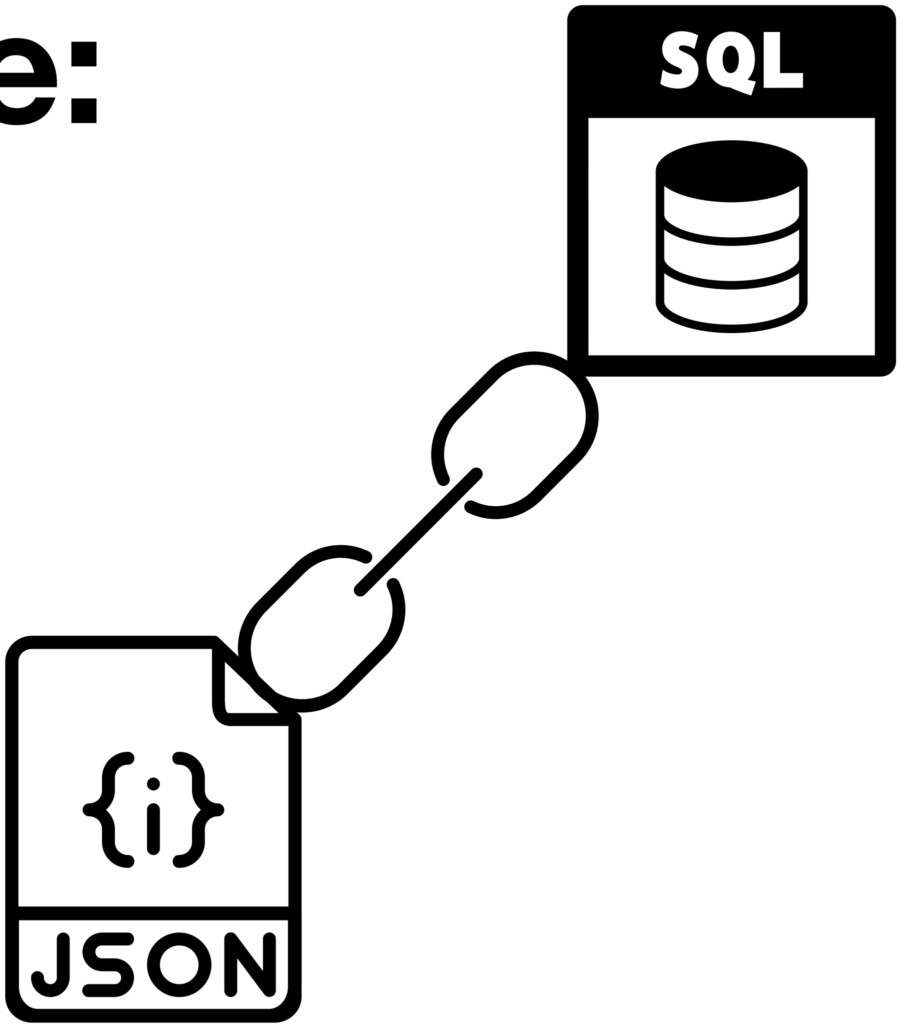
MySQL Document store: SQL and NoSQL united



Giuseppe Maxia

Vmware, Inc

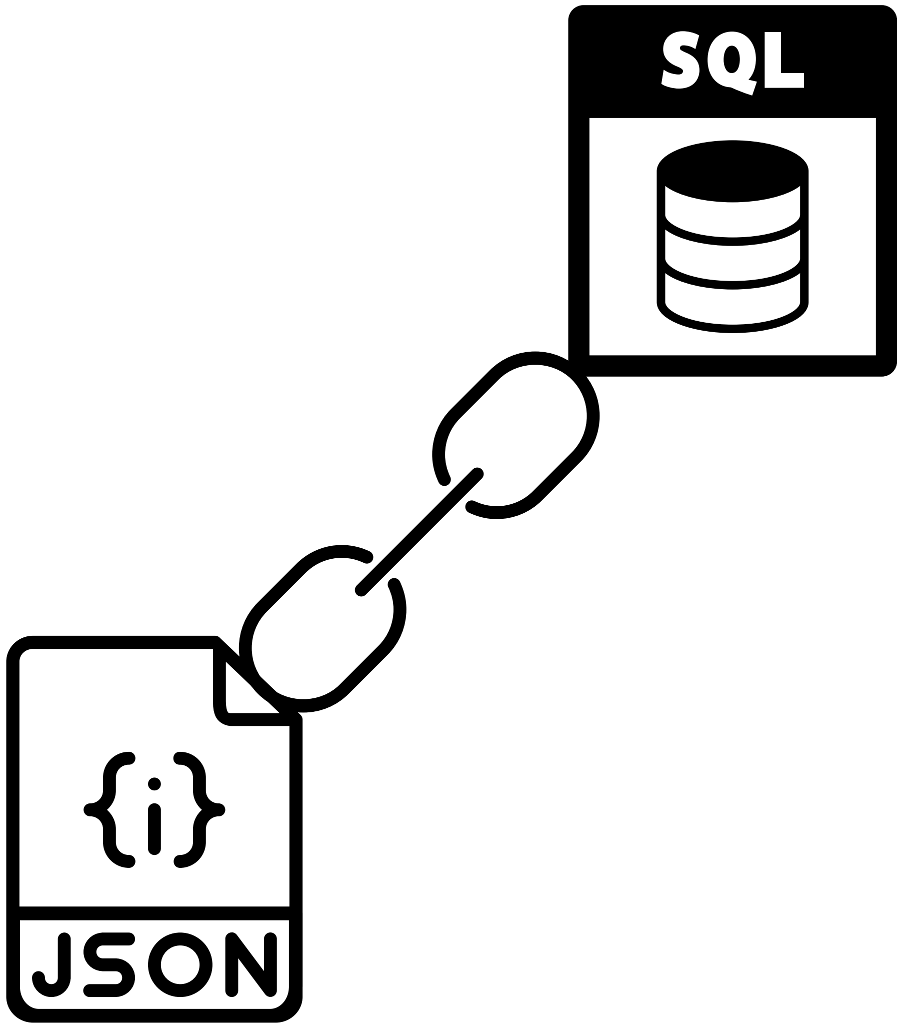
About me

Who's this guy?

- Giuseppe Maxia, a.k.a. "The Data Charmer"
- QA Architect at VMware
- 25+ years development and DB experience
- Long timer MySQL community member.
- Oracle ACE Director
- Blog: http://datacharmer.blogspot.com
- Twitter: @datacharmer

Agenda

- Document store in a nutshell
- X-Protocol overview
- X-Plugin installation
- MySQL shell installation
  - Using Docker
- Getting started
- Example: with the shell
- Example: data to and from MongoDB
- A look inside



Disclaimer

Better be clear about this

- **This is community work.**

- **Non affiliation:**
  - I don't work for Oracle. All I say here, good or bad, is my opinion.

- **Not talking for my company:**
  - All I say is my own stuff. My company does not influence or censor what I say here.

Requirements

This technology does not work with every version

- **MySQL 5.7.12 or later (contains the X-Plugin)**
- **MySQL shell (separate product)**

X- Protocol overview

A new protocol to talk to MySQL

- extends and replaces the traditional client/ server protocol
- allows asynchronous communication to the server
- uses different API calls
  - Javascript
  - Python
  - C#
  - Java

Universal API

It should be easy to switch

MySQL Shell **JavaScript** Code

```
// Create a new collection
var myColl = db.createCollection('my_collection');
// Insert a document
myColl.add( { name: 'Sakila', age: 15 } ).execute();
// Insert several documents at once
myColl.add( [
{ name: 'Susanne', age: 24 },
{ name: 'Mike', age: 39 } ] ).execute();
```

Universal API

It looks really easy to switch!

MySQL Shell **Python** Code

```
#  Create a new collection
myColl = db.createCollection('my_collection')
#  Insert a document
myColl.add( { 'name': 'Sakila', 'age':15 } ).execute()
#  Insert several documents at once
myColl.add( [
{ 'name': 'Susanne', 'age': 24 },
{ 'name': 'Mike', 'age' : 39 } ] ).execute()
```

The document store is not in the server by default

- MySQL server does not include the X-protocol
- You need to install a plugin for this
- and you need the MySQL shell (new product) to use it

!! WARNING !!

The server is GA , but ...

- **The document store comes with MySQL 5.7.12+**

!! WARNING !!

The server is GA , but ...

- **The document store comes with MySQL 5.7.12+**
- **HOWEVER**

!! WARNING !!

The server is GA , but ...

- **The document store comes with MySQL 5.7.12+**
- **HOWEVER**
  - THE PLUGIN IS **NOT** GA quality

!! WARNING !!

The server is GA , but ...

- **The document store comes with MySQL 5.7.12+**
- **HOWEVER**
  - THE PLUGIN IS **NOT** GA quality
  - It is, actually, pretty much alpha software

!! WARNING !!

The server is GA , but ...

- **The document store comes with MySQL 5.7.12+**
- **HOWEVER**
  - THE PLUGIN IS **NOT** GA quality
  - It is, actually, pretty much alpha software
- **DO NOT use it in production**

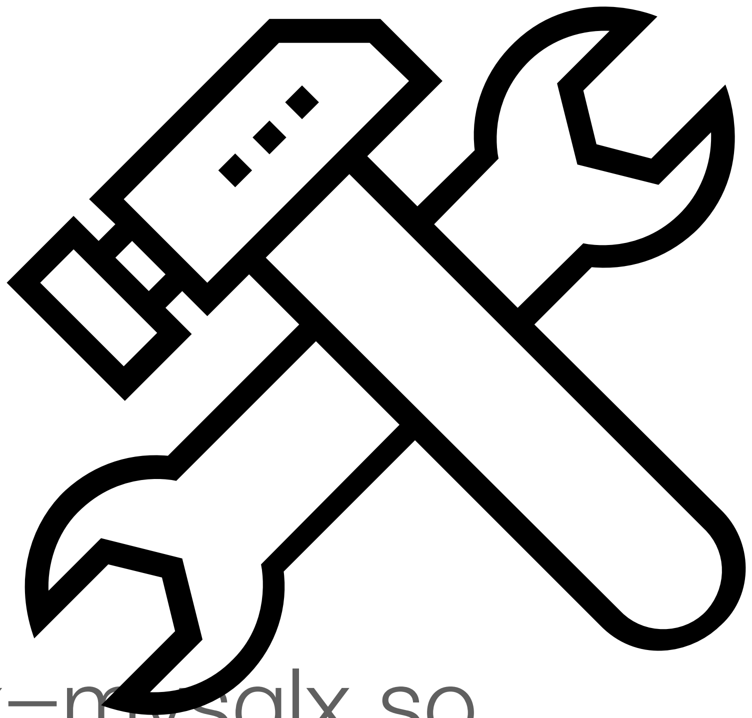
!! WARNING !!

The server is GA , but ...

- **The document store comes with MySQL 5.7.12+**
- **HOWEVER**
    - THE PLUGIN IS **NOT** GA quality
    - It is, actually, pretty much alpha software
- **DO NOT use it in production**

X-Plugin installation

The plugin comes with the server, but you need to enable it



- **Three methods:**
  - with mysqlsh
  - at startup, using --plugin-load=mysqlx=mysqlx.so
  - in SQL, using INSTALL PLUGIN

Method 1 : with mysqlsh

Using the mysql shell itself

```
mysqlsh \
    --classic \
    --user=msandbox \
    --password=msandbox \
    --port=5714 \
    --host=127.0.0.1 \
    --dba enableXProtocol
```

Method 2 : at startup

When we start the server

```
mysqld [...] --plugin-load=mysqlx=mysqlx.so
```

```
# or in the configuration file
[mysqld]
# ...
plugin-load=mysqlx=mysqlx.so
```

Method 3 : in SQL

At any moment

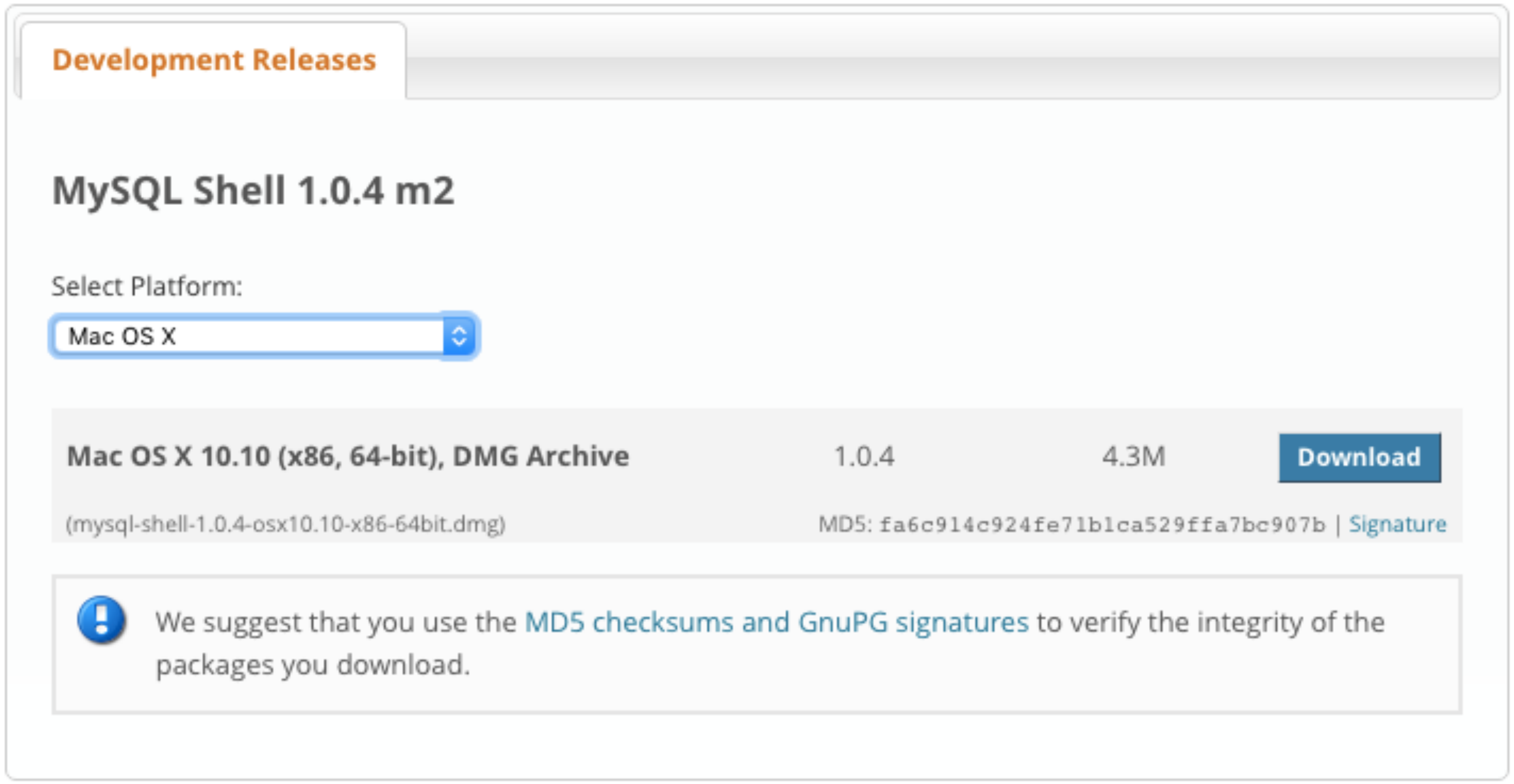
```
install plugin mysqlx soname 'mysqlx.so';
```

Gotchas

- X-Plugin listens to port 33060
- When you install with method 1, you use port 3306
- Afterwards, you use port 33060

MySQL Shell installation

You need the new client to use the new features

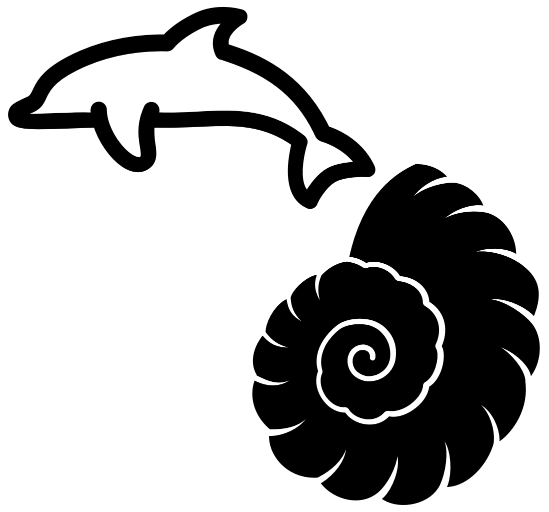


MySQL Shell 1.0.4 m2

Select Platform: Mac OS X

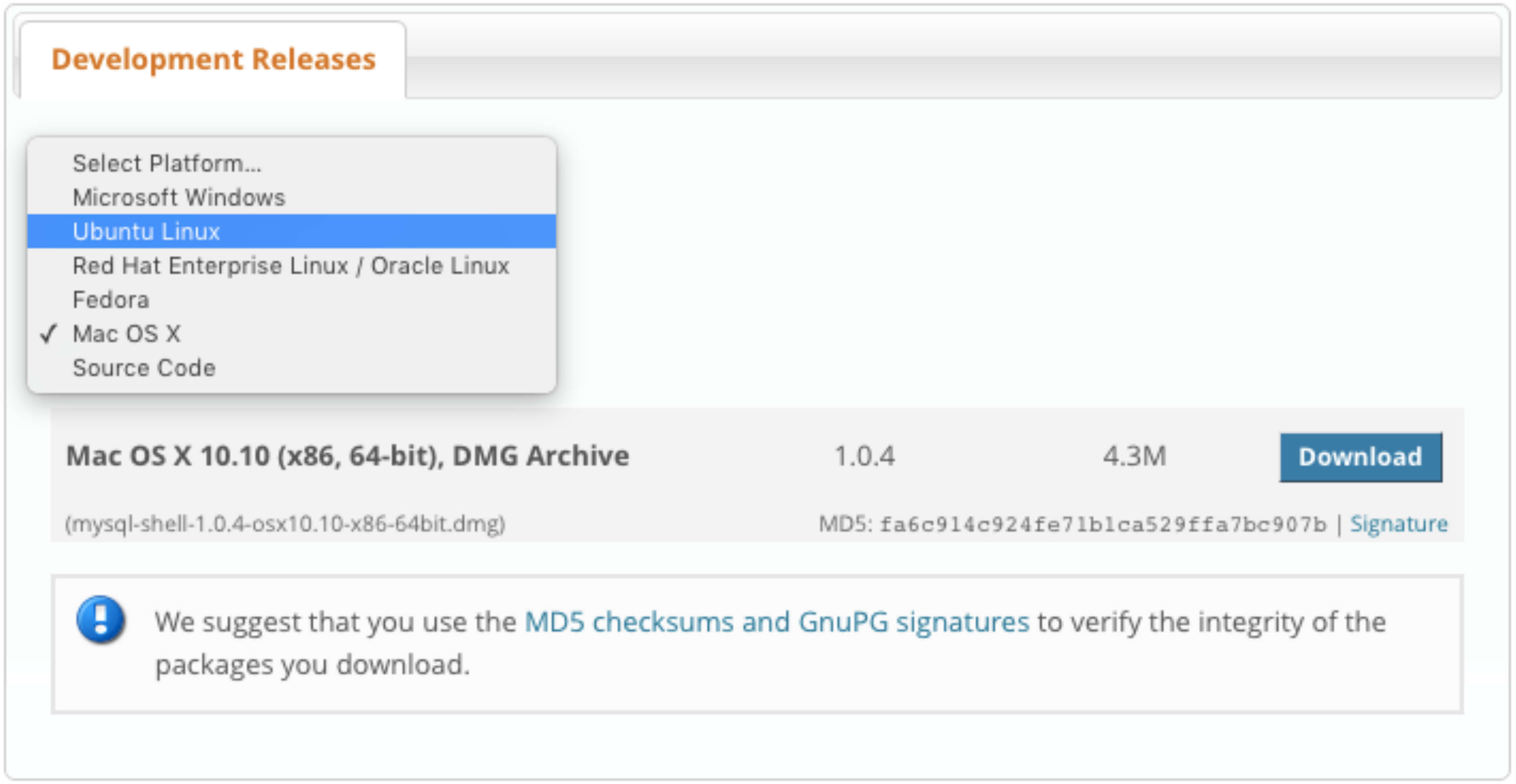
Mac OS X 10.10 (x86, 64-bit), DMG Archive	1.0.4	4.3M	Download
(mysql-shell-1.0.4-osx10.10-x86-64bit.dmg)	MD5: fa6c914c924fe71b1ca529ffa7bc907b | Signature		

We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.



MySQL Shell installation

You need the new client to use the new features



Mac OS X 10.10 (x86, 64-bit), DMG Archive 1.0.4 4.3M

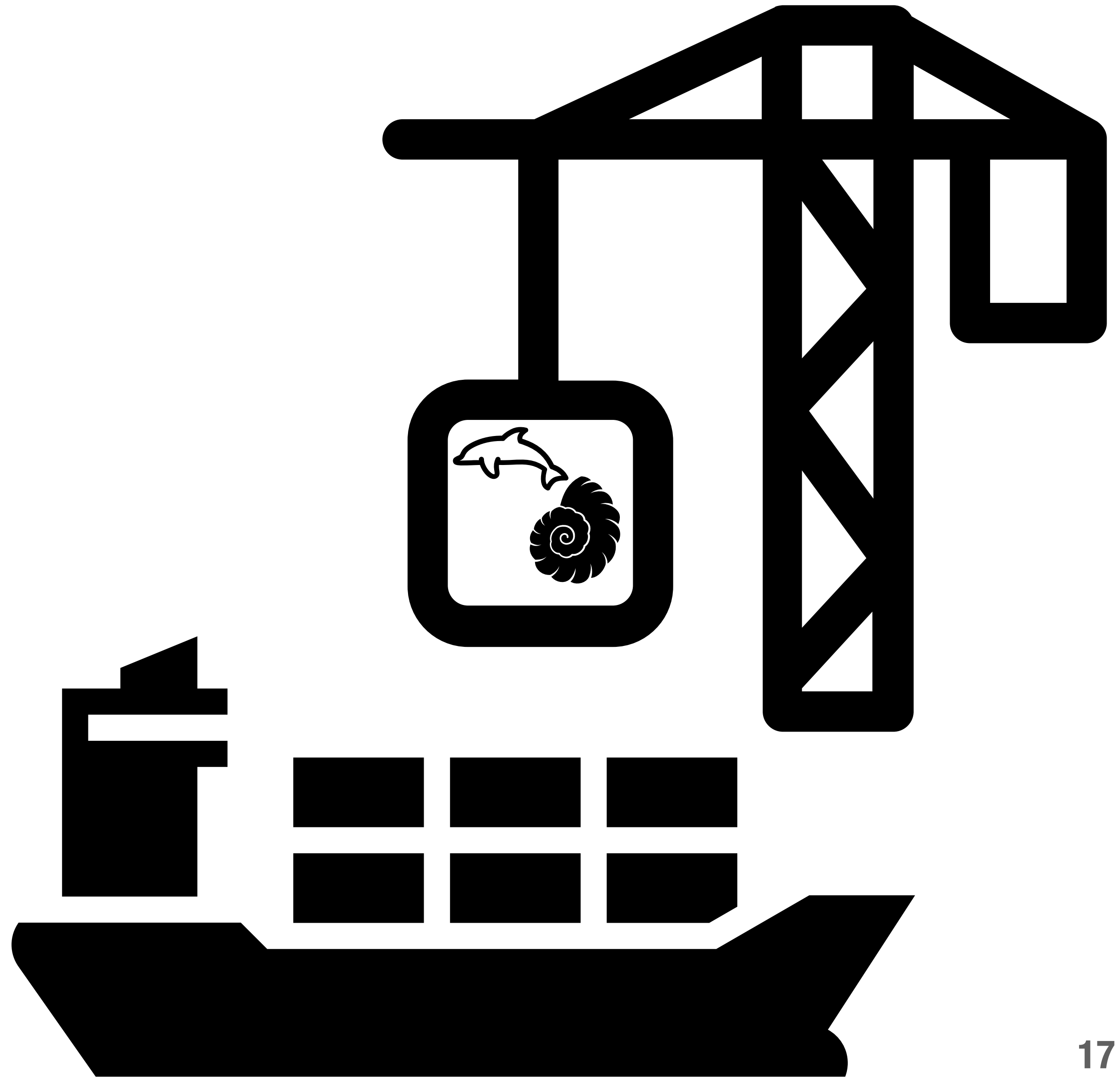
(mysql-shell-1.0.4-osx10.10-x86-64bit.dmg) MD5: fa6c914c924fe71b1ca529ffa7bc907b | Signature

We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

Shell with Docker

Instead of installing ...

- Using a Docker image
- Shell ready to use
- No side effects



Using Docker

No installation needed

```
$ docker network create mynet

$ docker run --name mybox --net mynet \
  -e MYSQL_ROOT_PASSWORD=secret \
  -d mysql/mysql-server

## WAIT 15 seconds
$ docker exec -ti mybox mysql -psecret \
    -e "install plugin mysqlx soname 'mysqlx.so'"

## LOAD SOMETHING

$ docker run --rm -it --net mynet \
    mysql/shell -u root -h mybox -p
```

Getting started

Let's practice with real data

- **Install MySQL 5.7.14**
- **load plugin**
- **Download the world_x database**
  - https://dev.mysql.com/doc/index-other.html
- **load the database**
- **connect using mysql shell**

Examples with the shell

Getting ready

```
make_sandbox 5.7.14 -- --load_plugin=mysqlx \
-c general_log=1
[...]
Your sandbox server was installed in $HOME/sandboxes/
msb_5_7_14

sudo netstat -atn | grep LISTEN | grep '5714\|33060'
tcp4       0      0  *.33060                *.*                    LISTEN
tcp4       0      0  127.0.0.1.5714         *.*                    LISTEN

 ~/sandboxes/msb_5_7_14/use \
    < ~/data/world_x-db/world_x.sql
```

As seen from the old client

Some things have two faces

```
~/sandboxes/msb_5_7_14/use world_x

mysql [localhost] {msandbox} (world_x) > show tables;
+-------------------+
| Tables_in_world_x |
+-------------------+
| City              |
| Country           |
| CountryInfo       |
| CountryLanguage   |
+-------------------+
4 rows in set (0.00 sec)
```

And from the new client (1)

Welcome to the machine!

```
$ mysqlsh -u msandbox -p  world_x
Creating an X Session to msandbox@localhost:33060/
world_x
Enter password:
Default schema `world_x` accessible through db.

Welcome to MySQL Shell 1.0.3 Development Preview

[...]

Type '\help', '\h' or '\?' for help.

Currently in JavaScript mode. Use \sql to switch to
SQL mode and execute queries.
mysql-js>
```

And from the new client (2)

Welcome to the machine!

```
mysql-js> db.getTables()
{
    "City": <Table:City>,
    "Country": <Table:Country>,
    "CountryLanguage": <Table:CountryLanguage>
}
mysql-js> db.getCollections()
{
    "CountryInfo": <Collection:CountryInfo>
}
mysql-js>
```

And from the new client (3)

This syntax does not work anymore!

```
// only works in mysqlsh 1.0.3

mysql-js> db.tables
{
    "City": <Table:City>,
    "Country": <Table:Country>,
    "CountryLanguage": <Table:CountryLanguage>
}
mysql-js> db.collections
{
    "CountryInfo": <Collection:CountryInfo>
}
mysql-js>
```

Starting something new

Schema-less!

```
mysql-js> nc=db.createCollection('person')
<Collection:person>
mysql-js>
mysql-js> db.getCollections()
{
    "CountryInfo": <Collection:CountryInfo>,
    "person": <Collection:person>
}
mysql-js>
```

Inserting data

REALLY schema-less!

```
mysql-js> nc.add({ name: "Joe", city: "Paris"})
Query OK, 1 item affected (0.00 sec)

mysql-js> nc.add({ name: "Frank", where_are_you_from:
"London"})
Query OK, 1 item affected (0.01 sec)
```

Retrieving data

This reminds me of something ...

```
mysql-js> nc.find()
[
    {
        "_id": "6eee6f07ab66e611564dfeeead98f1ef",
        "name": "Frank",
        "where_are_you_from": "London"
    },
    {
        "_id": "94b470f7aa66e611564dfeeead98f1ef",
        "city": "Paris",
        "name": "Joe"
    }
]
2 documents in set (0.00 sec)
```

Back to the old side

The general log shows what we were doing

```
CREATE TABLE `world_x`.`person` (doc JSON,_id
VARCHAR(32) GENERATED ALWAYS AS
(JSON_UNQUOTE(JSON_EXTRACT(doc, '$._id'))) STORED
PRIMARY KEY) CHARSET utf8mb4 ENGINE=InnoDB

 Query INSERT INTO `world_x`.`person` (doc) VALUES
('{\"_id\":\"94b470f7aa66e611564dfeeead98f1ef\",\"city
\":\"Paris\",\"name\":\"Joe\"}')

Query INSERT INTO `world_x`.`person` (doc) VALUES
('{\"_id\":\"6eee6f07ab66e611564dfeeead98f1ef\",\"name
\":\"Frank\",\"where_are_you_from\":\"London\"}')
```

A bigger collection

The world_x database comes with some beefy data

```
mysql-js> db.getCollections()
{
    "CountryInfo": <Collection:CountryInfo>,
    "person": <Collection:person>
}

mysql-js> ci=db.getCollection('CountryInfo')
<Collection:CountryInfo>
```

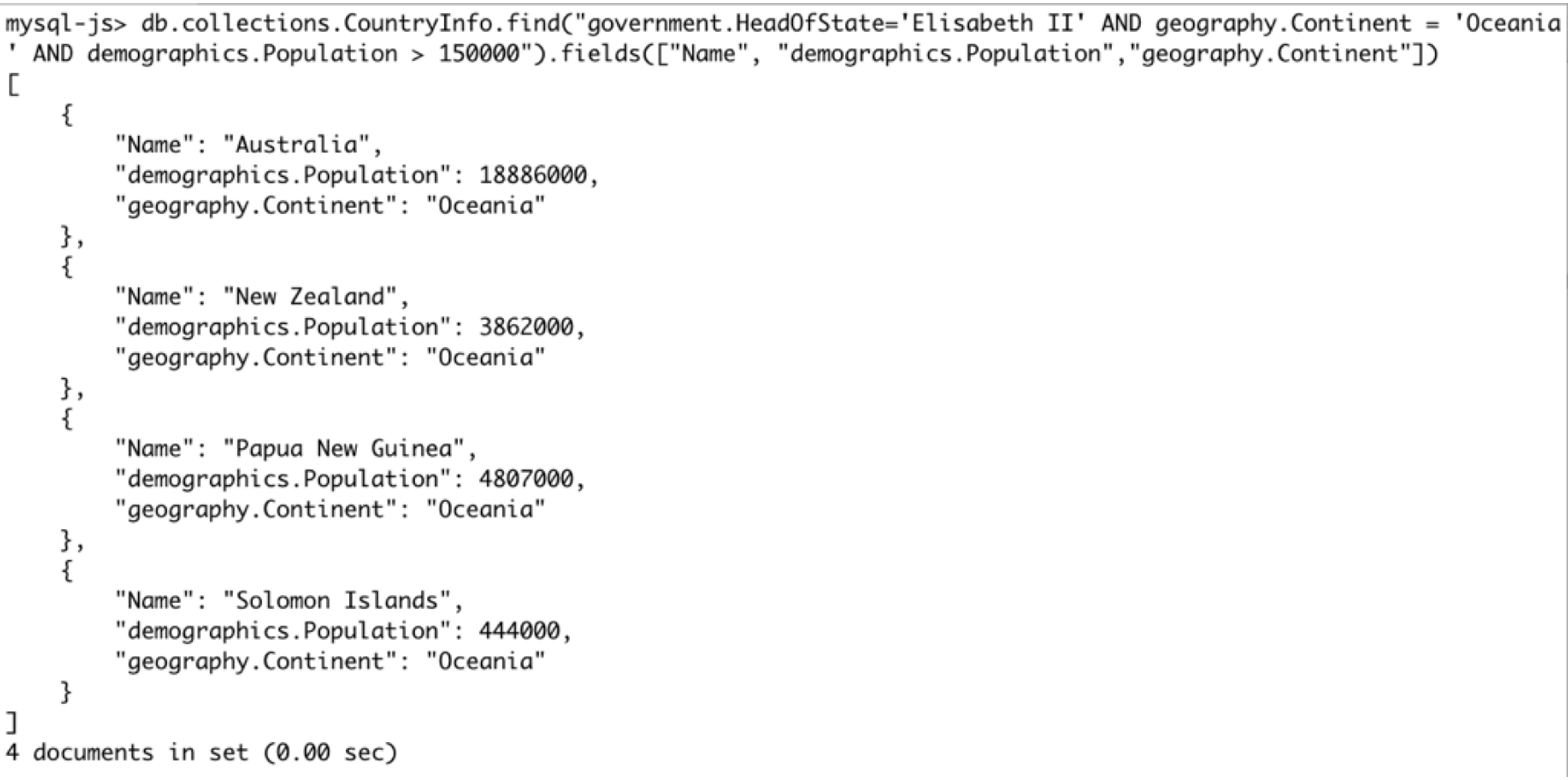
Sample data rom world_x

The data is in layers

```
mysql-js> ci.find().limit(1)
[
    {
        "GNP": 828,
        "IndepYear": null,
        "Name": "Aruba",
        "_id": "ABW",
        "demographics": {
            "LifeExpectancy": 78.4000015258789,
            "Population": 103000
        },
        "geography": {
            "Continent": "North America",
            "Region": "Caribbean",
            "SurfaceArea": 193
        },
        "government": {
            "GovernmentForm": "Nonmetropolitan Territory of The Netherlands",
            "HeadOfState": "Beatrix"
        }
    }
]
```

Complex queries are possible

Not always easy to get



```
mysql-js> db.collections.CountryInfo.find("government.HeadOfState='Elisabeth II' AND geography.Continent = 'Oceania
' AND demographics.Population > 150000").fields(["Name", "demographics.Population","geography.Continent"])
[
    {
        "Name": "Australia",
        "demographics.Population": 18886000,
        "geography.Continent": "Oceania"
    },
    {
        "Name": "New Zealand",
        "demographics.Population": 3862000,
        "geography.Continent": "Oceania"
    },
    {
        "Name": "Papua New Guinea",
        "demographics.Population": 4807000,
        "geography.Continent": "Oceania"
    },
    {
        "Name": "Solomon Islands",
        "demographics.Population": 444000,
        "geography.Continent": "Oceania"
    }
]
4 documents in set (0.00 sec)
```

Examples: to and from MongoDB

Since they are both schema-less ...

- **From MySQL to MongoDB**
  - extract data from a document store
  - feed it to MongoDB
- **From MongoDB to MySQL**
  - create collection
  - extract data
  - filter off the oddities
  - feed it to MySQL shell

From MySQL to Mongoldb (1)

Extracting data

```
// extract.js
var mysqlx = require('mysqlx').mysqlx;
var mySession =
mysqlx.getSession('msandbox:msandbox@127.0.0.1');
var result =
mySession.world_x.CountryInfo.find().execute();
var record = result.fetchOne();
while(record){
      print(record);
        record = result.fetchOne();
}

$ mysqlsh < extract.js > ~/data/country_info.json
```

From MySQL to Mongoldb (2)

import data to mongoldb

```
mongoimport --db test --collection countries \
    --drop --file /data/country_info.json
```

From MongoDB to MySQL (1)

First create the collection

```
mysql-js> db.createCollection('restaurants')
```

From MongoDB to MySQL (2)

Export the data from MongoDB

```
docker exec -ti mongo mongo --quiet \
    --eval 'DBQuery.shellBatchSize=300; var
all=db.restaurants.find() ; all' \
    | perl -pe 's/(?:ObjectId|ISODate)\(("[^"]+")\)/
$1/g' \
    > all_recs.json
```

Why do we need to filter

There is data like this:

```
{
        "_id" : ObjectId("57b81d385957bb0d60511ce5"),
        "borough" : "Bronx",
        "cuisine" : "Bakery",
        "grades" : [
           {
              "date" : ISODate("2014-03-03T00:00:00Z"),
              "grade" : "A",
              "score" : 2
           },
        ],
        "name" : "Morris Park Bake Shop",
        "restaurant_id" : "30075445"
}
```

From MongoDB to MySQL (3)

Importing into MySQL

```
grep -v '^Type' all_recs.json | \
  perl -ple 's/\r//;s/\Q$_/db.restaurants.add( $_ );/'
> t.txt
cat t.txt |
  mysqlsh -u msandbox --password=msandbox -i full test
```

```
t.txt contains lines like:

db.restaurants.add( { "_id" :
"57b81d385957bb0d60511ce5", "address" : { "building" :
"1007", "coord" : [ -73.856077, 40.848447 ],
"street" : "Morris Park Ave", "zipcode" : "10462" },
"borough" : "Bronx", "cuisine" : "Bakery", "grades" :
[ { "date" : "2014-03-03T00:00:00Z", "grade" : "A",
"score" : 2 }, { "date" : "2013-09-11T00:00:00Z",
"grade" : "A", "score" : 6 }, { "date" :
"2013-01-24T00:00:00Z", "grade" : "A", "score" : 10 },
{ "date" : "2011-11-23T00:00:00Z", "grade" : "A",
"score" : 9 }, { "date" : "2011-03-10T00:00:00Z",
"grade" : "B", "score" : 14 } ], "name" : "Morris Park
Bake Shop", "restaurant_id" : "30075445" } );
```

A look inside

What's a MySQL Document?

- **mysqlsh calls it a "collection"**
- **mysql calls it a table**
    - with a GENERATED _**id** field
    - with a json field

mysql

The old client view

```
show tables;
+--------------------+
| Tables_in_world_x  |
+--------------------+
| City               |
| Country            |
| CountryInfo        |
| CountryLanguage    |
+--------------------+
4 rows in set (0.00 sec)
```

mysqlsh

The document store view

```
mysql-js> db.getCollections()
{
    "CountryInfo": <Collection:CountryInfo>
}
mysql-js> db.getTables()
{
    "City": <Table:City>,
    "Country": <Table:Country>,
    "CountryLanguage": <Table:CountryLanguage>
}
```

mysql

The old client view

```
show create table CountryInfo\G
******* 1. row ***************************
       Table: CountryInfo
Create Table: CREATE TABLE `CountryInfo` (
  `doc` json DEFAULT NULL,
  `_id` varchar(32) GENERATED ALWAYS AS
(json_unquote(json_extract(`doc`,'$._id'))) STORED
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

general log

The "truth"

```
SELECT C.table_name AS name,
IF(ANY_VALUE(T.table_type)='VIEW', 'VIEW', IF(COUNT(*)
= COUNT(CASE WHEN (column_name = 'doc' AND data_type =
'json') THEN 1 ELSE NULL END) + COUNT(CASE WHEN
(column_name = '_id' AND generation_expression =
'json_unquote(json_extract(`doc`,''$._id''))') THEN 1
ELSE NULL END) + COUNT(CASE WHEN (column_name != '_id'
AND generation_expression RLIKE '^(json_unquote[[.
(.]])?json_extract[[.(.]]`doc`,''[[.$.]]([[...]]
[^[:space:][...]]+)+''[[.).]]{1,2}$') THEN 1 ELSE NULL
END), 'COLLECTION', 'TABLE')) AS type FROM
information_schema.columns AS C LEFT JOIN
information_schema.tables AS T USING (table_name)WHERE
C.table_schema = 'world_x' GROUP BY C.table_name ORDER
BY C.table_name
```

how x-plugin finds "collections"

I'd say it needs more integration

```
SELECT C.table_name AS name, IF(ANY_VALUE(T.table_type)='VIEW', 'VIEW',
IF(COUNT(*) = COUNT(CASE WHEN (column_name = 'doc' AND data_type = 'json') THEN 1
ELSE NULL END) + COUNT(CASE WHEN (column_name = '_id' AND generation_expression =
'json_unquote(json_extract(`doc`,''$._id''))') THEN 1 ELSE NULL END) + COUNT(CASE
WHEN (column_name != '_id' AND generation_expression RLIKE '^(json_unquote[[.
(.]])?json_extract[[.(.]]`doc`,''[[.$.]]([[...]][^[:space:][...]]+)+''[[.).]]
{1,2}$') THEN 1 ELSE NULL END), 'COLLECTION', 'TABLE')) AS type FROM
information_schema.columns AS C LEFT JOIN information_schema.tables AS T USING
(table_name)WHERE C.table_schema = 'world_x' GROUP BY C.table_name ORDER BY
C.table_name
```

```
+-----------------+------------+
| name            | type       |
+-----------------+------------+
| City            | TABLE      |
| Country         | TABLE      |
| CountryInfo     | COLLECTION |
| CountryLanguage | TABLE      |
+-----------------+------------+
```

Q & A