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)
How DBAs see data

This is not intuitive
How developers see data

This is driven by most programming languages

```json
{
    "name": "Joe",
    "dept": "Dev",
    "titles": [
        { "name": "Junior developer", "from": "2006-02-01", "until": "2010-10-31" },
        { "name": "Developer", "from": "2010-11-01", "until": "2012-11-30" },
        { "name": "Team Lead", "from": "2015-12-01", "until": "2014-11-30" }
    ],
    "salaries": [
        { "salary": 5000, "from": "2006-02-01", "until": "2008-06-30" },
        { "salary": 5100, "from": "2008-07-01", "until": "2010-06-30" },
        { "salary": 5200, "from": "2010-07-01", "until": "2012-06-30" },
        { "salary": 5300, "from": "2011-07-01", "until": "2013-06-30" },
        { "salary": 5400, "from": "2012-07-01", "until": "2014-06-30" }
    ]
}
```
DBAs vs. developers

It's a clash of data structures

```javascript
let xxxx = {
  x: {
    y: {
      z: [zzzz, zzzzz, zzzz],
      y: [aaaaa, yyyyyy, yyyyyy, yyyyyy],
    },
    z: [zzzz, zzzzz, zzzz],
  },
  y: [aaaa],
};
```
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

```javascript
// 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

```python
# 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 (separate product) to use it
!! WARNING !!
The server is GA, but ...

- The document store comes with MySQL 5.7.12+
- HOWEVER
  - The tools ARE NOT GA quality
  - They are, actually, pretty much alpha software
- Be careful when using it in production
"Using MySQL as a document store is currently a preproduction feature"
Readiness (as of April 2017)
There are several components to the document store

<table>
<thead>
<tr>
<th>MySQL SERVER 5.7.12+</th>
<th>PLUGIN</th>
<th>SHELL 1.0.8</th>
<th>CONNECTORS</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA</td>
<td>GA</td>
<td>RC</td>
<td>Alpha</td>
</tr>
</tbody>
</table>

- node.js
- java
- javascript
- python
- .net
- c++
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=3306 \
  --host=127.0.0.1 \
  --dba enableXProtocol
```
Method 2 : at startup

When we start the server

```
mysqld [...] --plugin-load=mysqlx=mysqlx.so \ 
   --mysqlx-port=15000

# or in the configuration file
[mysqld]
# ...
plugin-load=mysqlx=mysqlx.so
mysqlx-port=15000
```
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

<table>
<thead>
<tr>
<th>Development Releases</th>
</tr>
</thead>
</table>

**MySQL Shell 1.0.8 rc**

Select Operating System:

- **Mac OS X**

---

**Packages for Sierra (10.12) are compatible with El Capitan (10.11)**

<table>
<thead>
<tr>
<th>Mac OS X 10.12 (x86, 64-bit), DMG Archive</th>
<th>1.0.8</th>
<th>5.0M</th>
<th>Download</th>
</tr>
</thead>
<tbody>
<tr>
<td>(mysql-shell-1.0.8-rc-macos10.12-x86-64bit.dmg)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MD5: ab47029fed09bf2ef7fb49f4a2b37d0</td>
<td>Signature</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mac OS X 10.12 (x86, 64-bit), Compressed TAR Archive</th>
<th>1.0.8</th>
<th>4.8M</th>
<th>Download</th>
</tr>
</thead>
<tbody>
<tr>
<td>(mysql-shell-1.0.8-rc-macos10.12-x86-64bit.tar.gz)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MD5: 7ddb7f09de411080ce9ec22266598cbe</td>
<td>Signature</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

There are packages for most operating systems

- Select Operating System...
- Microsoft Windows
- Ubuntu Linux
- SUSE Linux Enterprise Server
- Red Hat Enterprise Linux / Oracle Linux
- Fedora
- Linux - Generic
- Mac OS X
- Source Code

- Linux - Generic (glibc 2.12) (x86, 64-bit), Compressed TAR Archive
  (mysql-shell-1.0.8-rc-linux-glibc2.12-x86-64bit.tar.gz)
  MD5: 14691e9c64f07c8425f4469443b6608b | Signature
  Size: 6.3M

- Linux - Generic (glibc 2.12) (x86, 32-bit), Compressed TAR Archive
  (mysql-shell-1.0.8-rc-linux-glibc2.12-x86-32bit.tar.gz)
  MD5: f3e68f6ea286786b1a7777cb9c5bc736c | Signature
  Size: 6.4M

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 ...

MySQL shell image not updated since 11 months ago

‣ Using a Docker image
‣ Shell ready to use
‣ No side effects
However ...

Using the latest mysqlsh release, there is a workaround:

docker run -d \
  -v $PWD/mysql-shell/bin/mysqlsh:/usr/bin/mysqlsh \
  -v ~/data:/data \
  --name mybox \
  -e MYSQL_ROOT_PASSWORD=secret \
  mysql/mysql-server

docker run -ti mybox bash
Getting started
Let's practice with real data

- Install MySQL 5.7.18
- 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 5.7.17 -- --load_plugin=mysqlx \
-c general_log=1
[...]
#Sandbox server installed in $HOME/sandboxes/msb_5_7_17

sudo netstat -atn | grep LISTEN | grep '5714\|33060'
#tcp4    0    0   *.33060       *.*       LISTEN
#tcp4    0    0  127.0.0.1.57147  *.*       LISTEN

$HOME/sandboxes/msb_5_7_17/use \
< $HOME/data/world_x-db/world_x.sql
```
As seen from the old client
Some things have two faces

~/sandboxes/msb_5_7_17/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)
Welcome to the machine!

And from the new client (1)

Welcome to the machine!

$ mysqlsh \

    --uri msandbox:msandbox@127.0.0.1:33060/world_x

Creating a Session to 'msandbox@127.0.0.1:33060/world_x'

Node Session successfully established. Default schema `mysql_x` accessible through db.
Welcome to MySQL Shell 1.0.8-rc

[…]

Type '\help', '\h' or '\?' for help, type '\quit' or '\q' to exit.

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!

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

Schema-less!

```javascript
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-j> `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":"94b470f7aa66e611564dfeeeead98f1ef","city":"Paris","name":"Joe"}')

Query INSERT INTO `world_x`.`person` (doc) VALUES ('{"_id":"6eee6f07ab66e611564dfeeeead98f1ef","name":"Frank","where_are_you_from":"London"}')
A complex example
A classical organization with sales

- departments
- categories
- person
- locations
- countries
- class
- sales
- customers
SQL to JSON (1)

The classical organization as a SQL result

```sql
select name, sale_date, sale_amount as '$$$', customer, class_name
from person
  inner join sales using (person_id)
  inner join customers using (customer_id)
  inner join class c on (c.class_id = sales.class_id);
```

+--------+------------+-------+-------------+------------+
| name   | sale_date  | $$$   | customer    | class_name |
+--------+------------+-------+-------------+------------+
| Frank  | 2003-10-01 | 23000 | DataSmart   | software   |
| Frank  | 2003-10-12 | 45000 | NewHardware | hardware   |
| Frank  | 2003-11-04 | 60000 | DataSmart   | services   |
| Susan  | 2003-11-02 | 25000 | ViewData    | software   |
| Susan  | 2003-10-13 | 18000 | SmartEdu    | services   |
| Martin | 2003-10-12 | 50000 | SmartEdu    | hardware   |
+--------+------------+-------+-------------+------------+
The classical organization as a collection of documents

```json
{
  "_id": 3,
  "country": "Germany",
  "category": "employee",
  "location": "Bonn",
  "name": "Frank",
  "salary": 5000,
  "gender": "m",
  "department": "sales",
  "sales": [
    {
      "class_name": "software",
      "sale_date": "2003-10-01",
      "customer": "DataSmart",
      "sale_amount": 23000
    },
    {
      "class_name": "hardware",
      "sale_date": "2003-10-12",
      "customer": "NewHardware",
      "sale_amount": 45000
    },
    {
      "class_name": "services",
      "sale_date": "2003-11-04",
      "customer": "DataSmart",
      "sale_amount": 60000
    }
  ]
}
```
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 from world_x

The data is in layers

```javascript
mysql-j> 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


[  
  {    
    "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
// extract.py
import mysqlx
import json

session = mysqlx.get_session( ... )
schema = session.get_schema('world_x')
collection = schema.get_collection('CountryInfo')
result = collection.find().execute()
docs = result.fetch_all()

for doc in docs:
    doc = dict(doc)
    print(json.dumps(doc, indent=4))

session.close()

# python extract.py > /data/country_info.json
From MySQL to Mongoldb (2)

import data to mongoldb

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

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

There is data like this:

```json
{
"_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"
}
```
Importing into MySQL

```python
schema = session.get_schema('test')
collection = schema.create_collection('restaurants')

with open('all_recs.json', 'r') as json_data:
    for line in json_data:
        skip=re.match('Type', line)
        if not skip:
            rec = json.loads(line)
            collection.add(rec).execute()
```

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
### show tables;

```sql
+-------------------+
| 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>
}
```
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`,'$.id'))) STORED  
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```
SELECT C.table_name AS name,
IF(ANY_VALUE(T.table_type)='VIEW', 'VIEW', IF(COUNT(*) = COUNT(CASE WHEN (column_name = 'doc' AND data_type = 'json') THEN 1 ELSE NULL END) + COUNT(CASE WHEN (column_name = '_id' AND generation_expression = 'json_unquote(json_extract(`doc`,''$_id''))') THEN 1 ELSE NULL END) + COUNT(CASE WHEN (column_name != '_id' AND generation_expression RLIKE '^\[\[.*\]\]$') 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
I'd say it needs more integration

```sql
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      |
More with mysql shell
A few tricks that could be useful

- Using mysqlsh to export in JSON format from regular MySQL tables
- Running mysqlsh in Docker without a dedicated container.
Examples
where to find the examples used in this presentation

https://github.com/datacharmer/mysql-document-store
Q & A