MySQL Document Store

Top 10 Reasons

A MySQL® White Paper

June, 2018

Table of Contents

Introduction

One of the great new features in MySQL 8.0 is the Document Store. JSON documents can be stored in collections and managed using CRUD operations. Instead of a mix of MongoDB and MySQL, now you can eliminate MongoDB and consolidate with MySQL !

MySQL can be used for NoSQL and SQL in same database server!

To use MySQL 8.0 as Document Store, the X plugin needs to be installed (by default since 8.0.11). This plugin enables the X DevAPI that provides a modern programming interface. Clients that communicate with a MySQL Server using the X Protocol can use the X DevAPI to develop applications. The X Protocol provides more flexible connectivity between those clients and the server. It supports both SQL and NoSQL APIs and can perform non-blocking asynchronous calls.

Of course, all Connectors provided by MySQL as well as the very popular PHP driver are compatible with the new X protocol.

Lets look at the top 10 reasons you should consider using NoSQL with MySQL:

1. MySQL cares about your data ! NoSQL full ACID compliant
2. CRUD operations (SQL is not mandatory anymore)
3. Schemaless
4. Documents can have the benefit of Data Integrity
5. Allows SQL (very important for analytics)
6. Eliminate16MB limitation for Document
7. Simple query syntax
8. Security
9. Simplify your DB infrastructure
10. MySQL DBAs already knows how to manage/tune/scale MySQL

and more…

- Lots of instrumentation
- Makes ORM obsolete
- MySQL Shell

1. MySQL cares about your data! NoSQL fully ACID compliant

Unlike traditional NoSQL databases, MySQL is fully ACID (Atomicity, Consistency, Isolation and Durability) compliant. NoSQL databases often lack **Durability**. This means that data can be lost after a crash. **Consistency** is also not guaranteed. As the MySQL Document Store relies on the InnoDB Storage Engine, the Document Store benefits from InnoDB's strength & robustness. By default, out-of-the-box, InnoDB is fully Durable and once data is acknowledged as committed, it won't be lost:

- `innodb_flush_log_at_trx_commit = 1`
- `innodb_doublewrite = ON`
- `sync_binlog = 1`

MySQL Document Store also supports **Atomicity** and **Isolation** as transactions are a core capability of InnoDB.

See example below:

```
MySQL [localhost+ ssl/docstore] JS> db.users.find()
[
    {
        "_id": "00005ad754c90000000000000001",
        "name": "lefred"
    }
]
1 document in set (0.0109 sec)

MySQL [localhost+ ssl/docstore] JS> session.startTransaction()
Query OK, 0 rows affected (0.0069 sec)

MySQL [localhost+ ssl/docstore] JS> db.users.add({name: 'dim0'})
Query OK, 1 item affected (0.0442 sec)

MySQL [localhost+ ssl/docstore] JS> db.users.find().fields('name')
[
    {
        "name": "lefred"
    },
    {
        "name": "dim0"
    }
]
2 documents in set (0.0579 sec)

MySQL [localhost+ ssl/docstore] JS> session.rollback()
Query OK, 0 rows affected (0.1201 sec)

MySQL [localhost+ ssl/docstore] JS> db.users.find().fields('name')
[
    {
        "name": "lefred"
    }
]
1 document in set (0.0004 sec)
```

As the example illustrates, transactions are supported in MySQL.

2. CRUD operations (SQL is not mandatory anymore)

With the new MySQL Document Store and the X protocol, new operations have been introduced to manage collections and/or relational tables. Those operations are called CRUD (Create, Read, Update, Delete) operations for data management without writing a single line of SQL.

Of course, CRUD operation have been differentiated into 2 groups, one to operate on *Collections* and one to operate on *Tables*:

CRUD functions on Collection	CRUD functions on Tables
`add()` : CollectionInsertObj	`insert()` : InsertObj
`find()` : CollectionFindObj	`select()` : SelectObj
`modify()` : CollectionUpdateObj	`update()` : UpdateObj
`remove()` : CollectionDeleteObj	`delete()` : DeleteObj

The above example illustrated how to use `find()` and `add()`.

3. Schemaless

MySQL Document Store enables the new universe of schemaless data. When storing documents, all the attributes don't need to be known in advance, the document can always be modified later. Since developers don't need to focus on table design, they don't need to take care of normalization, foreign keys and constraints or even datatypes. This allows for very quick initial development. Since documents are dynamic, schema migration is also not a problem, and large ALTER statements are not needed.

For example, see the following data:

```
MySQL [localhost+ ssl/docstore] JS> db.users.find()
[
    {
        "_id": "00005ad754c90000000000000001",
        "name": "lefred"
    },
    {
        "_id": "00005ad754c90000000000000003",
        "name": "dim0"
    },
    {
        "_id": "00005ad754c90000000000000004",
        "name": "Dave"
    },
    {
        "_id": "00005ad754c90000000000000005",
        "name": "Luis"
    },
    {
        "_id": "00005ad754c90000000000000006",
        "name": "Sunny"
    }
]
5 documents in set (0.0007 sec)
```

Imagine that developers want to add a new attribute for all the users. This can be achieved without having to run an ALTER statement:

```
MySQL [localhost+ ssl/docstore] JS> db.users.modify('1').set('department', 'development')
Query OK, 5 items affected (0.1910 sec)
```

Let's verify one of the records:

```
MySQL [localhost+ ssl/docstore] JS> db.users.find("name = 'Sunny'")
[
    {
        "_id": "00005ad754c90000000000000006",
        "department": "development",
        "name": "Sunny"
    }
]
1 document in set (0.0005 sec)
```

These capabilities give developers more freedom to create documents and they don't have to rely on an operational team to run large alter statements.

4. Documents with Data Integrity

Often it is desirable to combine the flexibility of schemaless with data integrity. With MySQL Document Store, constraints and foreign keys can be created and maintained for documents too.

This is an example where we have two collections: *users* and *departments*, a **GENERATED STORED** column was created to use as foreign key:

```
MySQL [localhost+ ssl/docstore] SQL> alter table departments add column dept varchar(20) gene
rated always as (doc->>"$.name") STORED ;
Query OK, 2 rows affected (0.3633 sec)

MySQL [localhost+ ssl/docstore] SQL> alter table users add column dept varchar(20) generated
always as (doc->>"$.department") STORED ;
Query OK, 5 rows affected (0.1302 sec)

MySQL [localhost+ ssl/docstore] SQL> select * from users;
+--------------------------------------------------------------------------------------------+----
------------------------+-------------+
| doc                                                                                        | _id
                         | dept        |
+--------------------------------------------------------------------------------------------+----
------------------------+-------------+
| {"_id": "00005ad754c90000000000000001", "name": "lefred", "department": "community"}  | 000
05ad754c90000000000000001 | community   |
| {"_id": "00005ad754c90000000000000003", "name": "dim0", "department": "community"}    | 000
05ad754c90000000000000003 | community   |
| {"_id": "00005ad754c90000000000000004", "name": "Dave", "department": "community"}    | 000
05ad754c90000000000000004 | community   |
| {"_id": "00005ad754c90000000000000005", "name": "Luis", "department": "development"}  | 000
05ad754c90000000000000005 | development |
| {"_id": "00005ad754c90000000000000006", "name": "Sunny", "department": "development"} | 000
05ad754c90000000000000006 | development |
+--------------------------------------------------------------------------------------------+----
------------------------+-------------+
5 rows in set (0.0010 sec)

MySQL [localhost+ ssl/docstore] SQL> select * from departments;
+--------------------------------------------------------------------------------------------+-------
----------------------+-------------+
| doc                                                                                          | _id
                       | dept        |
+--------------------------------------------------------------------------------------------+-------
----------------------+-------------+
| {"_id": "00005ad754c90000000000000007", "name": "development", "manager": "Tomas"} | 00005a
d754c90000000000000007 | development |
| {"_id": "00005ad754c90000000000000008", "name": "community", "manager": "Andrew"}  | 00005a
d754c90000000000000008 | community   |
+--------------------------------------------------------------------------------------------+-------
----------------------+-------------+
2 rows in set (0.0004 sec)
```

An index is added on these new columns:

```
MySQL [localhost+ ssl/docstore] SQL> alter table users add index dept_idx(dept);
Query OK, 0 rows affected (0.0537 sec)

MySQL [localhost+ ssl/docstore] SQL> alter table departments add index dept_idx(dept);
Query OK, 0 rows affected (0.1278 sec)
```

The constraint is created so that if a department is deleted, all users from that departments get removed:

```
MySQL [localhost+ ssl/docstore] SQL> alter table users add foreign key (dept) references depa
rtments(dept) on delete cascade;
Query OK, 5 rows affected (0.2401 sec)

MySQL [localhost+ ssl/docstore] SQL> delete from departments where doc->>"$.manager" like 'An
drew';
Query OK, 1 row affected (0.1301 sec)

MySQL [localhost+ ssl/docstore] SQL> select * from departments;
+--------------------------------------------------------------------------------------------+-------
----------------------+-------------+
| doc | _id | dept |
+--------------------------------------------------------------------------------------------+-------
----------------------+-------------+
| {"_id": "00005ad754c90000000000000007", "name": "development", "manager": "Tomas"} | 00005a
d754c90000000000000007 | development |
+--------------------------------------------------------------------------------------------+-------
----------------------+-------------+
1 row in set (0.0007 sec)

MySQL [localhost+ ssl/docstore] SQL> select * from users;
+--------------------------------------------------------------------------------------------+----
------------------------+-------------+
| doc | _id | dept |
+--------------------------------------------------------------------------------------------+----
------------------------+-------------+
| {"_id": "00005ad754c90000000000000005", "name": "Luis", "department": "development"} | 0000
5ad754c90000000000000005 | development |
| {"_id": "00005ad754c90000000000000006", "name": "Sunny", "department": "development"} | 000
05ad754c90000000000000006 | development |
+--------------------------------------------------------------------------------------------+----
------------------------+-------------+
2 rows in set (0.0006 sec)
```

As the example shows, it is very easy to implement foreign key constraints to enhance data integrity.

5. Allows SQL (very important for analytics)

As illustrated in the previous points, it's possible to mix SQL and NoSQL and go from one to the other and back. Sometimes good old SQL is very useful.

For example, front-end developers could just use NoSQL to create and consume data seen as objects and those working more in the back-office can keep using SQL to create reports and analytics.

To illustrate the power of MySQL with Documents, let's use the popular restaurant example collection from another popular NoSQL solution.

This is an example of document stored in the collection:

```
MySQL [localhost+ ssl/docstore] JS> db.restaurants.find().limit(1)
[
    {
        "_id": "5ad5b645f88c5bb8fe3fd337",
        "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"
    }
]
```

You can see that a restaurant can be graded and has a style of cuisine.

One easy query would be: What is the average grade for each restaurant ? (and limit the result to 10)

```
MySQL [localhost+ ssl/docstore] SQL> SELECT name, cuisine, avg(rating) FROM restaurants,
                                  -> JSON_TABLE(doc, "$" columns(name varchar(100) path "$.n
ame",
                                  ->            cuisine varchar(100) path "$.cuisine",
                                  ->            nested path "$.grades[*]"
                                  ->              columns (rating int path "$.score")))
                                  -> AS jt GROUP BY name, cuisine LIMIT 10;
+-------------------------------+---------------------------------+-------------+
| name                          | cuisine                         | avg(rating) |
+-------------------------------+---------------------------------+-------------+
| Morris Park Bake Shop         | Bakery                          |      8.2000 |
| Wendy'S                       | Hamburgers                      |      9.4404 |
| Dj Reynolds Pub And Restaurant | Irish                           |      9.2500 |
| Riviera Caterer               | American                        |      9.0000 |
| Tov Kosher Kitchen            | Jewish/Kosher                   |     17.7500 |
| Brunos On The Boulevard       | American                        |     17.0000 |
| Kosher Island                 | Jewish/Kosher                   |     10.5000 |
| Wilken'S Fine Food            | Delicatessen                    |     10.0000 |
| Regina Caterers               | American                        |      9.6000 |
| Taste The Tropics Ice Cream   | Ice Cream, Gelato, Yogurt, Ices |      8.2500 |
+-------------------------------+---------------------------------+-------------+
10 rows in set, 13 warnings (1.5114 sec)
```

MySQL Document Store can also leverage the power of Common Table Expression (CTEs).

How easily would a DBA answer this question in other Document Stores:

- Show the 10 best restaurants but they all must be from a different cuisine! (this means that if the two best restaurants are 2 Italian ones, only show the best of the two and the next restaurant in the list should be the third but doesn't serve Italian food).

In MySQL 8.0 Document Store this can be achieved with one query using a Common Table Expression (CTE):

```
MySQL [localhost+ ssl/docstore] SQL> WITH cte AS (SELECT doc->>"$.name" AS name,
                                  ->                    doc->>"$.cuisine" AS cuisine,
                                  ->  (SELECT AVG(score) FROM JSON_TABLE(doc, "$.grades[*]"
                                  ->   COLUMNS (score INT PATH "$.score")) AS r) AS avg_scor
e
                                  ->   FROM restaurants)
                                  ->  SELECT *, RANK() OVER (PARTITION BY cuisine ORDER BY
                                  ->  avg_score) AS `rank`
                                  -> FROM cte ORDER BY `rank`, avg_score DESC LIMIT 10;
+-----------------------------------------+------------------+-----------+------+
| name                                    | cuisine          | avg_score | rank |
+-----------------------------------------+------------------+-----------+------+
| Ravagh Persian Grill                    | Iranian          |   15.6667 |    1 |
| Camaradas El Barrio                     | Polynesian       |   14.6000 |    1 |
| Ellary'S Greens                         | Californian      |   12.0000 |    1 |
| General Assembly                        | Hawaiian         |   11.7500 |    1 |
| Catfish                                 | Cajun            |    9.0000 |    1 |
| Kopi Kopi                               | Indonesian       |    8.6667 |    1 |
| Hospoda                                 | Czech            |    7.8000 |    1 |
| Ihop                                    | Pancakes/Waffles |    7.2000 |    1 |
| New Fresco Toetillas Tommy'S Kitchen Inc | Chinese/Cuban    |    7.0000 |    1 |
| Snowdonia Pub                           | English          |    7.0000 |    1 |
+-----------------------------------------+------------------+-----------+------+
10 rows in set, 13 warnings (1.4284 sec)
```

So SQL is not mandatory anymore but it can still help. You can also do much more, like joining Documents with Relational Tables, etc… and of course don't forget the large panel of JSON functions our engineers have implemented.

6. Eliminate 16MB limitation for Documents

MongoDB users often complain that documents are limited to 16MB. There is no such limitation for MySQL Documents.

A single document in MySQL Document Store can have a size of **1GB** ! It's limited by the size of `max_allowed_packet`.

So if you have very large documents, MySQL is the best solution for you. In MySQL 8.0 we also improved how MySQL deals with such large documents. InnoDB implemented JSON partial updates and the replication team implemented Partial JSON for binary logs.

7. Simple query syntax

The goal of the new CRUD API is to provide developers an easy way to write applications without having to deal with the database backend and be able to query that data in the easiest way possible. Therefore, MySQL uses a very simple syntax to query documents stored in MySQL 8.0.

Let's compare the same query in MongoDB and MySQL:

First in MongoDB:

```
> db.restaurants.find({"cuisine": "French",
     "borough": { $not: /^Manhattan/} },
     {"_id":0, "name": 1,"cuisine": 1, "borough": 1}).limit(2)
{ "borough" : "Queens", "cuisine" : "French",
  "name" : "La Baraka Restaurant" }
{ "borough" : "Queens", "cuisine" : "French",
   "name" : "Air France Lounge" }
```

So what does that mean ? Check MySQL:

```
MySQL [localhost+ ssl/docstore] JS> restaurants.find("cuisine='French' AND borough!='Manhattan'").
fields(["name","cuisine","borough"]).limit(2)
[
    {
        "borough": "Queens",
        "cuisine": "French",
        "name": "La Baraka Restaurant"
    },
    {
        "borough": "Queens",
        "cuisine": "French",
        "name": "Air France Lounge"
    }
]
2 documents in set (0.0853 sec)
```

MySQL is much simpler!

8. Security

MySQL Document Store is secure by default: strong root password, password policy, roles, SSL. Several backups solutions are also available

When a MySQL instance is started, a SSL certificate will be created to communicate with the clients on a secure link. Also the super privilege user (root) will automatically have a strong password that will be stored in the error log and it must be changed at the first login. The new password will have to follow security rules that are also enabled by default.

```
2018-06-14T12:47:55.120634Z 5 [Note] [MY-010454] [Server] A temporary password is generated for ro
ot@localhost: (hdirvypB1iw
```

That password can be used to connect and can be set by the developer or DBA:

```
[root@mysql3 mysql]# mysqlsh root@127.0.0.1
Creating a session to 'root@127.0.0.1'
Enter password: ************
Fetching schema names for autocompletion... Press ^C to stop.
Error during auto-completion cache update: You must reset your password using ALTER USER statement
 before executing this statement.
Your MySQL connection id is 0 (X protocol)
No default schema selected; type \use  to set one.
MySQL Shell 8.0.11

Copyright (c) 2016, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type '\help' or '\?' for help; '\quit' to exit.

MySQL [127.0.0.1+] JS> session.sql('set password="MyV3ryStr0gP4ssw0rd%"')
Query OK, 0 rows affected (0.3567 sec)
```

Quit and login again and this time the prompt changed to notify SSL is being used. This can be easily verified:

```
[root@mysql3 mysql]# mysqlsh root@127.0.0.1
Creating a session to 'root@127.0.0.1'
Enter password: ****
Fetching schema names for autocompletion... Press ^C to stop.
Your MySQL connection id is 9 (X protocol)
Server version: 8.0.11 MySQL Community Server - GPL
No default schema selected; type \use  to set one.
MySQL Shell 8.0.11

Copyright (c) 2016, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type '\help' or '\?' for help; '\quit' to exit.


MySQL [127.0.0.1+ ssl] JS> \s
MySQL Shell version 8.0.11

Session type:                X
Connection Id:               9
Default schema:
Current schema:
Current user:                root@localhost
SSL:                         Cipher in use: DHE-RSA-AES128-GCM-SHA256 TLSv1.2
Using delimiter:             ;
Server version:              8.0.11 MySQL Community Server - GPL
Protocol version:            X protocol
Client library:              8.0.11
Connection:                  127.0.0.1 via TCP/IP
TCP port:                    33060
Server characterset:         utf8mb4
Schema characterset:         utf8mb4
Client characterset:         utf8mb4
Conn. characterset:          utf8mb4
Uptime:                      7 min 9.0000 sec
```

And if you are looking for other security features, please check MySQL Enterprise Edition.

9. Simplify your DB infrastructure

Instead of having to maintain multiple database products that often contain duplicate data, the MySQL Document Store enables organizations to consolidate RDBMSs and NoSQL.

Also, as already explained, you can mix Tables and Collections or run complex queries against your Documents. Additionally, an HA architecture can is also be easily implemented with MySQL InnoDB Cluster.

10. Your MySQL DBA already knows how to manage/tune/scale MySQL

Finally, if a DBA/OPS team already manages MySQL instances and you want to use NoSQL, simply use the CRUD operations of MySQL 8.0 and keep existing procedures in place to maintain your infrastructure. Same backups, same monitoring, same way to troubleshoot potential issues.

There is no need to invest in a new team or train an existing team to maintain yet another system.

MySQL 8.0 Document Store offers the best of both worlds and makes both developers and DBA/OPS happy.

Additional Resources

Documentation: Using MySQL as a Document Store
https://dev.mysql.com/doc/refman/8.0/en/document-store.html

Video: MySQL - Combining SQL and NoSQL
https://www.youtube.com/watch?v=yQvdJ2mVHjg&index=2&list=PLWx5a9Tn2EvGe-LGUpXYkv8-5LyKGh_DF

Blogs: MySQL Document Store
https://mysqlserverteam.com/category/docstore/

MySQL Customers and Case Studies
http://www.mysql.com/customers