Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes 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 upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.
Program Agenda

1. Introduction
2. The MySQL Document Store
3. Scale-Out - Sharding
4. Ease-of-Use – X DevAPI
5. Combining Document Store with Relational Model
6. Demo
Introduction
Document Oriented Databases

What is a Document?

- An object that can represent structured data
- Structure is implicit in the document; usually no external/central schema
- JSON (=JavaScript Object Notation)
  - Compact, popular and standardized
  - Can be represented natively in many languages (JavaScript, Python, etc.)
- Other popular encoding formats are XML, YAML etc

A JSON Document:

```json
{
  "_id": "AUT",
  "Name": "Austria",
  "GNP": 211860,
  "IndepYear": 1918,
  "demographics": {
    "LifeExpectancy": 77.699,
    "Population": 8091800
  },
  "geography": {
    "Continent": "Europe",
    "Region": "Western Europe",
    "SurfaceArea": 83859
  },
  "government": {
    "GovernmentForm": "Federal Republic",
    "HeadOfState": "Van der Bellen"
  }
}
```
Document Oriented Databases

Usability & Scalability

• **Schemaless**: No centralized database schema
  – Data model enforcement and validation (if any) at application layer
  – Simpler schema updates (no ALTER TABLE penalty)

• **NoSQL APIs**: Simpler programming interfaces
  – No specialized language for queries and data manipulation
  – Complex queries handled at application layer (no complex SELECTs, JOINs)
  – Document in, document out, manipulations at client side

• **Scalability**, but some drawbacks:
  – Limited database features (no foreign keys, no transactions, etc.)
  – Weak consistency guarantees
Why not...

- Have **both schema-less and schema** in the same technology stack?
- One that checks all the boxes of all stakeholders:

**Developers:**
- [x] Schemaless or/and Schema
- [x] Rapid Prototyping/Simpler APIs
- [x] Document Model
- [x] Transactions

**Operations:**
- [x] Performance Management/Visibility
- [x] Robust Replication, Backup, Restore
- [x] Comprehensive Tooling Ecosystem
- [x] Simpler application schema upgrades

**Business Owner:**
- [x] Don’t lose my data = ACID transactions
- [x] Capture all my data = Extensible/Schemaless
- [x] Products On Schedule/Time to Market = Rapid Development
The MySQL Document Store
What is the MySQL Document Store?

"An easy, straight forward way to work with JSON documents in MySQL"
What is the MySQL Document Store?

• Starting from MySQL 5.7
  – JSON SQL Datatype + JSON SQL functions

• But it is much more:
  – **Scale-Out** - A way to prepare applications for **massive scale-out**
    – First step to an out-of-the-box sharding solution
  – **Ease-of-Use** - A new approach for designing and writing MySQL database apps
    – A new querying interface called **DevAPI, based on CRUD**
      – `db.products.find("name like :n").bind("n", searchString).execute().fetch_all();`
    – Fast prototyping
    – No schema change headaches
Scaling the Document Store
Scaling MySQL – What is available today?

• Vertical Scaling (scaling a single machine instance) – **Available Today**
  – Big improvements in MySQL 5.7 and 8.0
  – 1M QPS
  – Multi-TB databases

• Read Scale-Out – **Available Today**
  – Already solved since more than 10 years
  – Big companies run hundreds or thousands of async read slaves
Scaling MySQL – Write Scale-Out

• **Myth:** *Relational databases don't scale for big data*

• **Truth:** Build your database using document model principles, and a RDBMS will scale as well!
  - Relationally designed databases are hard to scale **horizontally** (shard)
  - Foreign keys, transactional semantics, JOINs, strong global consistency, etc. ... make it difficult to partition the data across servers

• **MySQL Document Store will make it easy to build big scale databases**
  - Applications and database are designed in a way to simplify sharding
  - Certain features are avoided (or used carefully)
MySQL Write Scale-Out – 4 Steps

MySQL Document Store S1
Relational & Document Model

MySQL HA S2
Out-Of-Box HA

Read Scale-Out S3
Async Replication + Auto Failover

Write Scale-Out S4
Sharding

Timeline
MySQL Write Scale-Out – Step 4
MySQL InnoDB Cluster – Architecture

MySQL InnoDB cluster
How does it work?
How does the Document Store work?
Architecture from the Application’s POV

Frontend -> Backend
CRUD requests + JSON -> JSON
MySQL

Application
How does the Document Store work?

Architecture - Components

Application → Connector → X Plugin → MySQL

- DevAPI
- Protobuf / X Protocol / TCP/IP
- SQL
- InnoDB
Document Store DevAPI

• Commands serialized into Protobuf messages on the client side
• Transported via new "X Protocol" to the server
• Collections are stored as InnoDB tables
  – ACID compliance, transactions, replication, row locking etc all work as in plain MySQL
MySQL Document Store – Components

• MySQL X Plugin
  • Introduces X Protocol for relational- and document operations
  • Maps CRUD operations to standard SQL (relational tables, JSON datatype and functions)

• X Protocol
  • New MySQL client protocol based on top of industry standard (Protobuf)
  • Works for both, CRUD and SQL operations

• InnoDB Cluster
  • Read-Scaling, Write-Scaling, HA

• X DevAPI
  • New, modern, async developer API for CRUD and SQL operations on top of X Protocol
  • Introduces Collections as new Schema obj.

• MySQL Shell
  • Offers interactive X DevAPI mode for app prototyping

• MySQL Connectors
  • Support for X DevAPI for
    • JavaScript, Python, PHP, Java, C#, C++
DevAPI – CRUD Interface to the Document Store
Document Store DevAPI

Overview

• A Document-oriented database built on top of MySQL
• Native language API
  – Write queries and DB code directly in JavaScript, Python, C#, PHP, Java, etc.
• CRUD methods to insert, query, modify and delete JSON documents
• Relational database aspects are abstracted when working with documents
  – Dev focuses on Collections versus tables, columns, or schema
  – Just documents in collections
  – Simplified interface for indexing document fields
• ...but relational tables can also be used
Document Store DevAPI

Main Features

• Introducing the **Collection** Schema Object
  • Abstraction of a table for storing JSON Documents

• Modern API using method chaining
  • `db.products.find("name like :n").bind("n", searchString).execute().fetch_all();`

• **CRUD**
  • `.find()`, `.add()`, `.modify()`, `.remove()`

• Indexing, Transactions, Row Locking, ...
Example: Read

```python
# get a reference to the collection
prod = db.get_collection("products")
# execute query
q = prod.find("_id = :id").bind("id", path[1]).execute()
# fetch 1st result
doc = q.fetch_one()

if doc:
    self.response(200)
    self.send(str(doc)+"\n") # status code 200, with the document
else:
    self.response(404)
```
Example: Add

# Get the “products” Collection Object
prod = db.get_collection("products")

# get JSON data we want to add from HTTP request
data = self.rfile.read(int(self.headers.getheader('content-length')))

# Add JSON Document to the Collection
q = prod.add([data]).execute()

# Return the ID of the new Document to the Application via HTTP
self.send_result({'id': q.last_document_id}, 201)
Example: Delete

# Get the “products” Collection Object
prod = db.get_collection("products")

# Delete the object with ID that was requested
q = prod.remove("_id = :id").bind("id", path[1]).execute()

self.send_result({}, 200)
Example: Comparing with raw SQL...

```sql
SELECT JSON_OBJECT(
  'name', JSON_EXTRACT(doc,'$.name'),
  'zip', JSON_EXTRACT(doc, '$.address.zip'))
FROM `order`
WHERE (JSON_UNQUOTE(JSON_EXTRACT(doc,'$.address.zip')) IN
  ('91234','94231'));
```

```javascript
order.find("address.zip in ('91234', '94231')").
  patchFields({'name':'name', 'zip':'address.zip'});
```
Combining Document Store with Relational Model

SQL Interface to the Document Store
Document Store with SQL

• Available starting with MySQL 5.7
• JSON Datatype
• JSON Functions
• JSON Path Syntax
• JSON Indexing
• SQL Syntax Extensions
Document Store with SQL

JSON Datatype

- Store JSON data in table columns
- Validates format
- Internal binary format designed for fast lookups and partial updates
- Mix and match with SQL
- Convert (CAST) to and from string
Document Store with SQL

JSON Functions

• Construct JSON values
  – JSON_OBJECT('field', 'value', ...) → {"field": "value", ...}
  – JSON_ARRAY(1, 2, 3) → [1,2,3]
  – JSON_QUOTE('string')

• Query contents
  – JSON_EXTRACT('{"field": "value"}', '.field') → "value"
  – JSON_CONTAINS(['[1,2,3]', '3']) → 1 (true)
  – JSON_KEYS(), JSON_CONTAINS_PATH(), JSON_LENGTH() etc
Document Store with SQL

JSON Functions

• Modify JSON values
  – JSON_SET({'"name": "Alice"'}, '$.name', 'Bob') → {'"name": "Bob"'}
  – JSON_INSERT(), JSON_APPEND(), JSON_ARRAY_APPEND() etc

• Aggregate rows into arrays or objects
  – JSON_ARRAYAGG(), JSON_OBJECTAGG()
  – SELECT JSON_ARRAYAGG(name) FROM users
    → ['alice', 'bob', ...]
Document Store with SQL

JSON Path Syntax

• Refer to fields inside a JSON document
  
  { "field":
    { "array":
      [{"value": 123}]
    }
  }

  $.field.array[0].value

• Use in JSON functions
  – JSON_EXTRACT(document,'$.address.zip')

• Inline JSON Path Syntax to refer to JSON contents in SQL
  – SELECT doc->>'$.description' FROM products
Document Store with SQL

JSON Indexing

• Index on specific values inside JSON documents

• Virtual columns allow indexes on JSON fields
  – Create a virtual column to "look in" a JSON document
  – Create index on the virtual column

• Foreign keys can also be created on virtual columns
Document Store with SQL

EXAMPLE: Query JSON Objects from Table Columns

SELECT JSON_OBJECT('id', cu.id,
        'name', cu.name,
        'email', cu.email,
        'city', ci.city) as customer
FROM customer cu
JOIN city ci ON ci.id = cu.city_id
Demo – DevAPI in MySQL Shell
MySQL Document Store

Summary and Take Away

• New, modern way to develop database applications
• Combine best of relational and document oriented models
• MySQL InnoDB Cluster – Future proof for HA and scale-out deployments

• Blogs: mysqlserverteam.com/category/docstore/
Thank you!