MySQL 5.7 & JSON: New Opportunities for Developers

Manyi Lu

Director
MySQL Optimizer Team, Oracle
April, 2016

ORACLE

MySQL

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.

Why JSON Support in MySQL?

- Seemless integration of relational and schema-less data
- Leverage existing database infrastructure for new applications
- Provide a native JSON datatype
- Provide a set of built-in JSON functions

Agenda

- **JSON data type**
- JSON functions
- Indexing JSON data
- A real life example

The New JSON Datatype

```
CREATE TABLE employees (data JSON);
INSERT INTO employees VALUES
 ('{"id": 1, "name": "Jane"}'),
 ('{"id": 2, "name": "Joe"}');
SELECT * FROM employees;
+-----------------------------------+
| data                              |
+-----------------------------------+
| {"id": 1, "name": "Jane"}         |
| {"id": 2, "name": "Joe"}          |
+-----------------------------------+
```

- Validation on INSERT
- No reparsing on SELECT
- Optimized for read
- Dictionary of sorted keys
- Can compare JSON/SQL
- Can convert JSON/SQL
- Supports all native JSON datatypes
- Also supports date, time, timestamp etc.

Agenda

- JSON data type
- **JSON functions**
- Indexing JSON data
- A real life example

JSON Functions

- Info
  - JSON_VALID()
  - JSON_TYPE()
  - JSON_KEYS()
  - JSON_LENGTH()
  - JSON_DEPTH()
  - JSON_CONTAINS_PATH()
  - JSON_CONTAINS()
- Modify
  - JSON_REMOVE()
  - JSON_ARRAY_APPEND()
  - JSON_SET()
  - JSON_INSERT()
  - JSON_ARRAY_INSERT()
  - JSON_REPLACE()
- Create
  - JSON_MERGE()
  - JSON_ARRAY()
  - JSON_OBJECT()
- Get data
  - JSON_EXTRACT()
  - JSON_SEARCH()
- Helper
  - JSON_QUOTE()
  - JSON_UNQUOTE()

Inlined JSON Path Expressions

[[database.]table.]column->”$<path spec>”

```
SELECT * FROM employees WHERE data->”$.name” = “Jane";

Is a short hand for

SELECT * FROM employees WHERE JSON_EXTRACT(data, “$.name” ) = “Jane”;
```

- SELECT * FROM employees WHERE data->'$.id'= 2;
- ALTER … ADD COLUMN id INT AS (data->'$.id') …
- CREATE VIEW .. AS SELECT data->'$.id', data->'$.name' FROM …

Agenda

- JSON data type
- JSON functions
- **Indexing JSON data**
- A real life example

Generated Columns

```
CREATE TABLE order_lines
 (orderno integer,
  lineno integer,
  price decimal(10,2),
  qty   integer,
  sum_price decimal(10,2) GENERATED ALWAYS AS (qty * price) STORED );
```

- Column generated from the expression
- VIRTUAL: computed when read, not stored, indexable
- STORED: computed when inserted/updated, stored in SE, indexable
- Useful for:
  - Functional index
  - Materialized cache for complex conditions
  - Simplify query expression

Functional Index

```
CREATE TABLE order_lines
 (orderno integer,
  lineno integer,
  price decimal(10,2),
  qty   integer,
  sum_price decimal(10,2) GENERATED ALWAYS AS (qty * price) VIRTUAL);

ALTER TABLE order_lines ADD INDEX  idx (sum_price);
```

- ***Online*** index creation
- Composite index on a mix of ordinary, virtual and stored columns

Indexing JSON data

```
CREATE TABLE employees (data JSON);

ALTER TABLE employees
ADD COLUMN name VARCHAR(30) AS (JSON_UNQUOTE(data->”$.name”)) VIRTUAL,
ADD INDEX name_idx (name);
```

- Functional index approach
- Use inlined JSON path or JSON_EXTRACT to specify field to be indexed
- Support both virtual and stored generated columns

Generated column: STORED vs VIRTUAL

	Pros	Cons
STORED	• Fast retrieval	• Require table rebuild at creation • Update table data at INSERT/UPDATE • Require more storage space
VIRTUAL	• Metadata change only, instant • Faster INSERT/UPDATE, no change to table	• Compute when read, slower retrival

Indexing Generated Column: STORED vs VIRTUAL

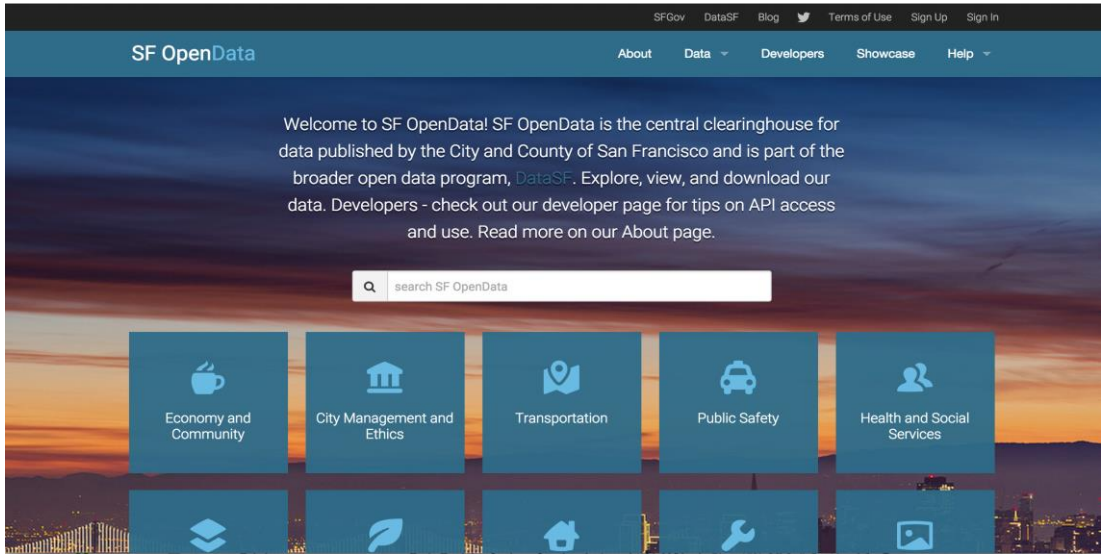
	Pros	Cons
STORED	• Primary & secondary index • B-TREE, Full text, R-TREE • Independent of SE • Online operation	• Duplication of data in base table and index
VIRTUAL	• Less storage • Online operation	• Secondary index only • B-TREE only • Require SE support

Agenda

- JSON data type
- JSON functions
- Indexing JSON data
- **A real life example**

Using Real Life Data

- Via **SF OpenData**
- 206K JSON objects representing subdivision parcels.




```
CREATE TABLE features (
   id INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,
   feature JSON NOT NULL
);
```

- Imported from https://github.com/zemirco/sf-city-lots-json + small tweaks

```
{
  "type":"Feature",
  "geometry":{
    "type":"Polygon",
    "coordinates":[
      [
        [-122.42200352825247,37.80848009696725,0],
        [-122.42207601332528,37.808835019815085,0],
        [-122.42110217434865,37.808803534992904,0],
        [-122.42106256906727,37.80860105681814,0],
        [-122.42200352825247,37.80848009696725,0]
      ]
    ]
  },
  "properties":{
    "TO_ST":"0",
    "BLKLOT":"0001001",
    "STREET":"UNKNOWN",
    "FROM_ST":"0",
    "LOT_NUM":"001",
    "ST_TYPE":null,
    "ODD_EVEN":"E",
    "BLOCK_NUM":"0001",
    "MAPBLKLOT":"0001001"
  }
}
```


Naive Performance Comparison

Unindexed traversal of 206K documents

```
# as JSON type
SELECT DISTINCT
 feature->"$.type" as json_extract
FROM features;
+-----------------+
| json_extract |
+-----------------+
| "Feature"     |
+-----------------+
1 row in set (1.25 sec)
```

```
# as TEXT type
SELECT DISTINCT
 feature->"$.type" as json_extract
FROM features;
+-----------------+
| json_extract |
+-----------------+
| "Feature"      |
+-----------------+
1 row in set (12.85 sec)
```

Explanation: Binary format of JSON type is very efficient at searching. Storing as TEXT performs over 10x worse at traversal.

Create Index

From table scan on 206K documents to index scan on 206K materialized values

```
ALTER TABLE features ADD feature_type VARCHAR(30) AS (feature->"$.type") VIRTUAL;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

ALTER TABLE features ADD INDEX (feature_type);
Query OK, 0 rows affected (0.73 sec)
Records: 0  Duplicates: 0  Warnings: 0

SELECT DISTINCT feature_type FROM features;
+------------------+
| feature_type |
+------------------+
| "Feature"      |
+------------------+
1 row in set (0.06 sec)
```

Meta data change only (FAST). Does not need to touch table.

Creates index online. Does not modify table rows.

Down from 1.25 sec to 0.06 sec

JSON Path Search

- Locate scalar values inside a document
- Provides a novice way to know the path. To retrieve via: *[[database.]table.]column->"$<path spec>"*

```
SELECT JSON_SEARCH(feature,'one', 'MARKET')
AS extract_path FROM features
WHERE id = 121254;

+-----------------------------+
| extract_path                |
+-----------------------------+
| "$.properties.STREET" |
+-----------------------------+
1 row in set (0.00 sec)
```

```
SELECT feature->"$.properties.STREET"
 AS property_street FROM features
WHERE id = 121254;

+----------------------+
| property_street |
+----------------------+
| "MARKET"          |
+----------------------+
1 row in set (0.00 sec)
```

JSON Array Creation

```
SELECT JSON_ARRAY(id,
 feature->"$.properties.STREET",
 feature->"$.type") AS json_array
FROM features ORDER BY RAND() LIMIT 3;

+-----------------------------------------+
| json_array                              |
+-----------------------------------------+
| [65298, "10TH", "Feature"]              |
| [122985, "08TH", "Feature"]             |
| [172884, "CURTIS", "Feature"]           |
+-----------------------------------------+
3 rows in set (2.66 sec)
```

Evaluates a (possibly empty) list of values and returns a JSON array containing those values

What is on Our Roadmap?

- Advanced JSON functions, e.g. JSON table function
- Multi-value index for efficient queries against array fields
- In-place update of JSON documents
- Full text and GIS index on virtual columns
- Improved performance through condition pushdown

Hardware and Software
Engineered to Work Together