MySQL 8.0: Document Store

How to Mix NoSQL & SQL in MySQL 8.0
MySQL 8.0 Document Store

How to Mix NoSQL & SQL in MySQL 8.0

Frédéric Descamps - MySQL Community Manager - Oracle
Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purpose 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 product remains at the sole discretion of Oracle.
about me - http://about.me/lefred

Who am I?
Frédéric Descamps

- @lefred
- MySQL Evangelist
- Hacking MySQL since 3.23
- devops believer
- living in Belgium BE
- http://lefred.be
Why?

Developers don’t really like SQL

SQL  NoSQL
Why?

SQL can be complicated and slows down the initial development
easy operations

What do developers want?
Use Objects / Documents

Developers want to just use objects (that’s why they usually love what DBAs hates: ORMs !).
Use Objects / Documents

Developers want to just use objects (that’s why they usually love what DBAs hates: ORMs!).

They want to deal with these objects easily (CRUD operations) and they don’t want to think about schema design (slows down the initial development process).
Use Objects / Documents

Developers want to just use objects (that’s why they usually love what DBAs hates: ORMs!).

They want to deal with these objects easily (CRUD operations) and they don’t want to think about schema design (slows down the initial development process).

But they also want to keep their data safe and use transactions.
ready for some fun?

Let’s start !
Migration from MongoDB to MySQL DS

For this example, I will use the well known collection:

```
[root@myserver1 ~]# mongoexport -c restaurants > from_mongo.json
connected to: 127.0.0.1
exported 25359 records
```
Migration from MongoDB to MySQL DS

For this example, I will use the well known collection:

```
[root@myserver1 ~]# mongoexport -c restaurants > from_mongo.json
connected to: 127.0.0.1
exported 25359 records
```

```
MySQL [docstore] on localhost:33060+ 2018-09-09 20:01:45  Threads running: 2
```

```
JSON util.importJson('/vagrant/from_mongo.json', {schema: 'docstore', collection: 'restaurants', convertBsonOid: true})
Importing from file "/vagrant/from_mongo.json" to collection `docstore`.<`restaurants` in MySQL Server at localhost
```

```
.. 25359.. 25359
Processed 15.90 MB in 25359 documents in 11.4145 sec (2.22K documents/s)
Total successfully imported documents 25359 (2.22K documents/s)
```
Let’s query

MySQL 127.0.0.1:33060 JS restaurants.find()
Let’s query

That’s too much records to show it here... let’s limit it
```json
[{
  "_id": "5943c83d1adc26055941640c",
  "address": {
    "building": "351",
    "coord": [
      -73.9851,
      40.7677
    ],
    "street": "West 57 Street",
    "zipcode": "10019"
  },
  "borough": "Manhattan",
  "cuisine": "Irish",
  "grades": [
    {
      "date": "2014-09-06T00:00:00Z",
      "grade": "A",
      "score": 2
    },
    {
      "date": "2013-07-22T00:00:00Z",
      "grade": "A",
      "score": 11
    },
    {
      "date": "2012-07-31T00:00:00Z",
      "grade": "A",
      "score": 12
    },
    {
      "date": "2011-12-29T00:00:00Z",
      "grade": "A",
      "score": 12
    }
  ],
  "name": "Dj Reynolds Pub And Restaurant",
  "restaurant_id": "30191841"
}]
```

1 document in set (0.08 sec)
Some more examples

```javascript
const restaurants = require('restaurants').restaurants;

const findQuery = restaurants.find().fields(['name', 'cuisine']).limit(2);

result = findQuery.execute();

console.log(result);

2 documents in set (0.00 sec)
```
Some more examples

---

Let's add a selection criteria:

```sql
MySQL> 127.0.0.1:33060> JS> restaurants.find().fields(["name","cuisine"]).limit(2)

[{
  "cuisine": "Irish",
  "name": "Dj Reynolds Pub And Restaurant"
},
{
  "cuisine": "American",
  "name": "Riviera Caterer"
}]
2 documents in set (0.00 sec)
```

---

```sql
MySQL> 127.0.0.1:33060> JS> restaurants.find("cuisine='Italian'").fields(["name","cuisine"]).limit(2)

[{
  "cuisine": "Italian",
  "name": "Philadelhia Grille Express"
},
{
  "cuisine": "Italian",
  "name": "Isle Of Capri Restaurant"
}]
2 documents in set (0.00 sec)
```
Using IN...

```sql
db.restaurants.find("cuisine in ('Italian', 'Spanish')")
    .fields('name', 'cuisine').limit(2)
```

```
[{
    "cuisine": "Italian",
    "name": "Marchis Restaurant"
},
{
    "cuisine": "Italian",
    "name": "Crystal Room"
}
]
```

2 documents in set (0.0021 sec)
And for developers?

**NoSQL + SQL = MySQL 8.0**

- search - add

<table>
<thead>
<tr>
<th>Name</th>
<th>Borough</th>
<th>Cuisine</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Park Pizzeria &amp; Restaurant</td>
<td>Queens</td>
<td>Pizza/Italian</td>
</tr>
<tr>
<td>Parkside Restaurant</td>
<td>Queens</td>
<td>Italian</td>
</tr>
<tr>
<td>Don Peppe</td>
<td>Queens</td>
<td>Italian</td>
</tr>
<tr>
<td>Cara Mia</td>
<td>Queens</td>
<td>Italian</td>
</tr>
<tr>
<td>Jack's Pizza &amp; Pasta</td>
<td>Queens</td>
<td>Pizza/Italian</td>
</tr>
<tr>
<td>Piccola Venezia</td>
<td>Queens</td>
<td>Italian</td>
</tr>
<tr>
<td>Amore Pizzeria &amp; Restaurant</td>
<td>Queens</td>
<td>Pizza/Italian</td>
</tr>
</tbody>
</table>

...
And for developers?

[Blank space]
And for developers?

Easy, using only CRUD operations!
MySQL Document Store Objects

- Schema
  - Browse Functions
    - getCollections()
    - getTables()
  - DbObject Instance Functions
    - getCollection() : CollectionObj
    - getTable() : TableObj
    - getCollectionAsTable() : TableObj
  - Create Functions
    - createCollection()

- Collection
  - CRUD Functions
    - add() : CollectionInsertObj
    - find() : CollectionFindObj
    - modify() : CollectionUpdateObj
    - remove() : CollectionDeleteObj
  - Index Functions
    - createIndex()
    - dropIndex()
    - getIndexes()
  - Document and Structure Functions
    - newDoc()
    - count() : Integer

- Table
  - Relational SQL CRUD Functions
    - insert() : InsertObj
    - select() : SelectObj
    - update() : UpdateObj
    - delete() : DeleteObj
  - Index Functions
    - dropIndex()
    - getIndexes()
  - General Functions
    - count() : Integer
CRUD operations

Add
CRUD operations

Add

Remove
CRUD operations

Modify
All you need to know is here:
we do care about your data

MySQL Document Store is Full ACID Compliant
Document Store Full ACID!

It relies on the proven MySQL InnoDB’s strength & robustness:
Document Store Full ACID!

It relies on the proven MySQL InnoDB’s strength & robustness:

-
Document Store Full ACID!

It relies on the proven MySQL InnoDB’s strength & robustness:

- 
- 

Copyright @ 2019 Oracle and/or its affiliates. All rights reserved.
Document Store Full ACID!

It relies on the proven MySQL InnoDB’s strength & robustness:

- 
- 
-
Document Store Full ACID!

It relies on the proven MySQL InnoDB’s strength & robustness:

- 
- 
- 
- 
Document Store Full ACID!

It relies on the proven MySQL InnoDB’s strength & robustness:

- 
- 
- 

We do care about your data!
MySQL DS – ACID

Transactions support

```sql
MySQL localhost:33060+ fred JS session.startTransaction()
Query OK, 0 rows affected (0.0006 sec)
MySQL localhost:33060+ fred JS test.add({name: 'the René'})
Query OK, 1 item affected (0.1661 sec)
MySQL localhost:33060+ fred JS test.find()
[
    {
        "_id": "00005ade551100000000000000000001",
        "name": "fred"
    },
    {
        "_id": "00005ade551100000000000000000002",
        "name": "the René"
    }
]
2 documents in set (0.0019 sec)
```
MySQL DS – ACID

Transactions support

```
MySQL> session.rollback()
Query OK, 0 rows affected (0.0992 sec)

MySQL> test.find()
[{
    "_id": "00005ade55110000000000000000001",
    "name": "fred"
}]
1 document in set (0.0130 sec)
```
OK we have Document Store, CRUD and ACID

but what makes MySQL Document Store unique?
Challenge: list the best restaurant of each type of food and show the top 10, with the best one first!

don’t forget that all these restaurants are just JSON documents
NoSQL as SQL – aggregation

```sql
WITH cte1 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_score
            FROM restaurants) SELECT *, RANK() OVER
            ( PARTITION BY cuisine ORDER BY avg_score DESC ) AS `rank`
            FROM cte1 ORDER BY `rank`, avg_score DESC LIMIT 10;
```

<table>
<thead>
<tr>
<th>name</th>
<th>cuisine</th>
<th>avg_score</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Juice It Health Bar</td>
<td>Juice, Smoothies, Fruit Salads</td>
<td>75.0000</td>
<td>1</td>
</tr>
<tr>
<td>Golden Dragon Cuisine</td>
<td>Chinese</td>
<td>73.0000</td>
<td>1</td>
</tr>
<tr>
<td>Palombo Pastry Shop</td>
<td>Bakery</td>
<td>69.0000</td>
<td>1</td>
</tr>
<tr>
<td>Go Go Curry</td>
<td>Japanese</td>
<td>65.0000</td>
<td>1</td>
</tr>
<tr>
<td>K &amp; D Internet Inc</td>
<td>Café/Coffee/Tea</td>
<td>61.0000</td>
<td>1</td>
</tr>
<tr>
<td>Koyla</td>
<td>Middle Eastern</td>
<td>61.0000</td>
<td>1</td>
</tr>
<tr>
<td>Ivory D O S Inc</td>
<td>Other</td>
<td>60.0000</td>
<td>1</td>
</tr>
<tr>
<td>Espace</td>
<td>American</td>
<td>56.0000</td>
<td>1</td>
</tr>
<tr>
<td>Tacos Al Suadero</td>
<td>Mexican</td>
<td>52.0000</td>
<td>1</td>
</tr>
<tr>
<td>Rose Pizza</td>
<td>Pizza</td>
<td>52.0000</td>
<td>1</td>
</tr>
</tbody>
</table>
NoSQL as SQL - aggregation

MySQL 8.0 Document Store - How to Mix NoSQL & SQL in MySQL 8.0

Common Table Expression (CTE)

WITH cte1 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_score
FROM restaurants) SELECT *
RANK() OVER
(PARTITION BY cuisine ORDER BY avg_score DESC) AS `rank`
FROM cte1 ORDER BY `rank` ASC;
NoSQL as SQL - aggregation

```sql
WITH cte1 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_score
        FROM restaurants)
SELECT *, RANK() OVER
  (PARTITION BY cuisine ORDER BY avg_score DESC) AS `rank`
FROM cte1 ORDER BY `rank`, avg_score DESC LIMIT 10;
```

Common Table Expression (CTE)

Window Function
NoSQL + SQL = MySQL 8.0

Top of the best restaurant for each type of cuisine

<table>
<thead>
<tr>
<th>Name</th>
<th>Cuisine</th>
<th>Average Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Juice It Health Bar</td>
<td>Juice, Smoothies, Fruit Salads</td>
<td>75.0000</td>
</tr>
<tr>
<td>Golden Dragon Cuisine</td>
<td>Chinese</td>
<td>73.0000</td>
</tr>
<tr>
<td>Palombo Pastry Shop</td>
<td>Bakery</td>
<td>69.0000</td>
</tr>
<tr>
<td>Go Go Curry</td>
<td>Japanese</td>
<td>65.0000</td>
</tr>
<tr>
<td>Koya</td>
<td>Middle Eastern</td>
<td>61.0000</td>
</tr>
<tr>
<td>K &amp; D Internet Inc</td>
<td>Café/Coffee/Tea</td>
<td>61.0000</td>
</tr>
<tr>
<td>Ivory D O S Inc</td>
<td>Other</td>
<td>60.0000</td>
</tr>
<tr>
<td>Espace</td>
<td>American</td>
<td>56.0000</td>
</tr>
<tr>
<td>Tacos Al Suadero</td>
<td>Mexican</td>
<td>52.0000</td>
</tr>
<tr>
<td>Rose Pizza</td>
<td>Pizza</td>
<td>52.0000</td>
</tr>
</tbody>
</table>
Virtual Columns, Indexes, ...

And for the DBA?
DS for DBAs: JSON

The DBAs can benefit from all JSON functions and capabilities.
DS for DBAs: JSON

The DBAs can benefit from all JSON functions and capabilities.

```sql
ALTER TABLE restaurants ADD COLUMN borough VARCHAR(20) GENERATED ALWAYS AS (json_unquote(json_extract(`doc`, '$.borough'))) VIRTUAL;
```
DS for DBAs: JSON

The DBAs can benefit from all JSON functions and capabilities.

```sql
ALTER TABLE restaurants ADD COLUMN borough VARCHAR(20) GENERATED ALWAYS AS (json_unquote(json_extract(`doc`, '$.borough'))) VIRTUAL;
```

same as:

```sql
ALTER TABLE restaurants ADD COLUMN borough VARCHAR(20) GENERATED ALWAYS AS (doc->>"$.borough") VIRTUAL;
```
DS for DBAs: JSON

The DBAs can benefit from all JSON functions and capabilities.

```sql
ALTER TABLE restaurants ADD COLUMN borough VARCHAR(20) GENERATED ALWAYS AS (json_unquote(json_extract('doc', '$.borough'))) VIRTUAL;
```

**same as:**

```sql
ALTER TABLE restaurants ADD COLUMN borough VARCHAR(20) GENERATED ALWAYS AS (doc->>'$.borough') VIRTUAL;
```

**we can use it in SQL:**

```sql
SELECT _id, borough FROM restaurants LIMIT 5;
```

<table>
<thead>
<tr>
<th>_id</th>
<th>borough</th>
</tr>
</thead>
<tbody>
<tr>
<td>59ca58986b977d7c822812b7</td>
<td>Bronx</td>
</tr>
<tr>
<td>59ca58986b977d7c822812b8</td>
<td>Brooklyn</td>
</tr>
<tr>
<td>59ca58986b977d7c822812b9</td>
<td>Manhattan</td>
</tr>
<tr>
<td>59ca58986b977d7c822812ba</td>
<td>Brooklyn</td>
</tr>
<tr>
<td>59ca58986b977d7c822812bb</td>
<td>Queens</td>
</tr>
</tbody>
</table>

5 rows in set (0.00 sec)
SQL and JSON: validation

```
MySQL> localhost:33060+ docstore SQL ALTER TABLE restaurants ADD COLUMN cuisine VARCHAR(20) GENERATED ALWAYS AS (doc->>"$.cuisine") VIRTUAL, WITH VALIDATION;
ERROR: 1406: Data too long for column 'cuisine' at row 10
```
SQL and JSON: validation

```sql
ALTER TABLE restaurants ADD COLUMN cuisine VARCHAR(20) GENERATED ALWAYS AS (doc->>'$.cuisine') VIRTUAL, WITH VALIDATION;
ERROR: 1406: Data too long for column 'cuisine' at row 10
```

```sql
ALTER TABLE restaurants ADD COLUMN cuisine VARCHAR(20) GENERATED ALWAYS AS (LEFT(doc->>'$.cuisine',20)) VIRTUAL, WITH VALIDATION;
Query OK, 25359 rows affected (0.00 sec)
```
SQL and JSON: validation

```
MySQL localhost:33060+ docstore SQL ALTER TABLE restaurants ADD COLUMN cuisine
VARCHAR(20) GENERATED ALWAYS AS (doc->>'$.cuisine') VIRTUAL, WITH VALIDATION;
ERROR: 1406: Data too long for column 'cuisine' at row 10
```

```
MySQL localhost:33060+ docstore SQL ALTER TABLE restaurants ADD COLUMN cuisine
VARCHAR(20) GENERATED ALWAYS AS (LEFT(doc->>'$.cuisine',20)) VIRTUAL, WITH VALIDATION;
Query OK, 25359 rows affected (0.00 sec)
```

```
MySQL localhost:33060+ docstore SQL SELECT _id, borough, cuisine FROM restaurants
LIMIT 5;
```

<table>
<thead>
<tr>
<th>_id</th>
<th>borough</th>
<th>cuisine</th>
</tr>
</thead>
<tbody>
<tr>
<td>59ca58986b977d7c822812b7</td>
<td>Bronx</td>
<td>Bakery</td>
</tr>
<tr>
<td>59ca58986b977d7c822812b8</td>
<td>Brooklyn</td>
<td>Hamburgers</td>
</tr>
<tr>
<td>59ca58986b977d7c822812b9</td>
<td>Manhattan</td>
<td>Irish</td>
</tr>
<tr>
<td>59ca58986b977d7c822812ba</td>
<td>Brooklyn</td>
<td>American</td>
</tr>
<tr>
<td>59ca58986b977d7c822812bb</td>
<td>Queens</td>
<td>Jewish/Kosher</td>
</tr>
</tbody>
</table>
SQL & JSON

But what for?
Example

```
EXPLAIN SELECT doc->>'$.name' AS name, cuisine, borough FROM restaurants WHERE cuisine='Italian' AND borough='Brooklyn' LIMIT 2

id: 1
select_type: SIMPLE
table: restaurants
partitions: NULL
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 24890
filtered: 1.0000001192092896
Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

Note (code 1003): /* select#1 */ select json_unquote(json_extract('docstore`restaurants`doc`$.name')) AS `name`, `docstore`restaurants`cuisine AS `cuisine`, `docstore`restaurants`borough AS `borough` from `docstore`restaurants where (`docstore`restaurants`borough = 'Brooklyn') and (`docstore`restaurants`cuisine` = 'Italian') limit 2
Example (2)

```
EXPLAIN
SELECT doc->>$_.name" AS name, cuisine, borough FROM restaurants WHERE cuisine='Italian' AND borough='Brooklyn' LIMIT 2

id: 1
select_type: SIMPLE
table: restaurants
partitions: NULL
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 24890
filtered: 1.0000001192092896
Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

Note (code 1003): /* select#1 */ select json_unquote(json_extract(`docstore`.`restaurants`.`doc`, '$.name')) AS `name`, `docstore`.`restaurants`.`cuisine` AS `cuisine`, `docstore`.`restaurants`.`borough` AS `borough` from `docstore`.`restaurants` where ((`docstore`.`restaurants` .`borough` = 'Brooklyn') and (`docstore`.`restaurants`.`cuisine` = 'Italian')) limit 2
Example (3) - Index

```sql
ALTER TABLE restaurants ADD INDEX cuisine_borough_idx(cuisine,borough);
Query OK, 0 rows affected (0.00 sec)

EXPLAIN SELECT doc->>'$.name' AS name, cuisine, borough
FROM restaurants WHERE cuisine='Italian' AND borough='Brooklyn' LIMIT 2;

id: 1
select_type: SIMPLE
table: restaurants
partitions: NULL
type: ref
possible_keys: cuisine_borough_idx
key: cuisine_borough_idx
key_len: 166
ref: const,const
rows: 192
filtered: 100
Extra: NULL
```

Note (code 1003): */ select#1 */ select json_unquote(json_extract('docstore'.restaurants.'doc','$name')) AS 'name', 'docstore'.restaurants.'cuisine' AS 'cuisine', 'docstore'.restaurants.'borough' AS 'borough' FROM 'docstore'.restaurants WHERE ('docstore'.restaurants.'borough' = 'Brooklyn') AND ('docstore'.restaurants.'cuisine' = 'Italian') limit 2
Example (4) - Index

```sql
ALTER TABLE restaurants ADD INDEX cuisine_borough_idx(cuisine,borough);
Query OK, 0 rows affected (0.00 sec)

EXPLAIN SELECT doc->>'$.name' AS name, cuisine, borough FROM restaurants WHERE cuisine='Italian' AND borough='Brooklyn' LIMIT 2
```

```sql
id: 1
select_type: SIMPLE
table: restaurants
partitions: NULL
type: ref
possible_keys: cuisine_borough_idx
key: cuisine_borough_idx
key_len: 166
ref: const, const
rows: 192
filtered: 100
Extra: NULL
```

1 row in set, 1 warning (0.00 sec)

Note (code 1003): /* select#1 */ select json_unquote(json_extract( 'docstore'.'.restaurants'.'.doc','$.name' )) AS 'name', 'docstore'.'.restaurants'.'.cuisine' AS 'cuisine', 'docstore'.'.restaurants'.'.borough' AS 'borough' FROM 'docstore'.'.restaurants' WHERE (( 'docstore'.'.restaurants'.'.borough' = 'Brooklyn') AND ( 'docstore'.'.restaurants'.'.cuisine' = 'Italian' )) limit 2
```
Indexes in Document Store

It’s also possible to create indexes without using SQL syntax:

```javascript
db.restaurants.createIndex('cuisine_idx', {
  fields:{field: "$\.cuisine", required: false, type: "text(20)"}
})
```
Indexes in Document Store

It’s also possible to create indexes without using SQL syntax:

```javascript
db.restaurants.createIndex('cuisine_idx',
  {fields:{field: "$ .cuisine", required: false, type: "text(20)"}})
```

```sql
show create table restaurants"
```

```
Table: restaurants
Create Table: CREATE TABLE `restaurants` (  `doc` json DEFAULT NULL,
  `_id` varbinary(32) GENERATED ALWAYS AS (json_unquote(json_extract(`doc`,_utf8mb4'$.id')))
) STORED NOT NULL,
  `$ix_t20_BC26D4DF1273E3F7412529AEE9E95A0CC8475CEB` text GENERATED ALWAYS AS (json_unquote(json_extract(`doc`,_utf8mb4'.$.cuisine')))) VIRTUAL,
  PRIMARY KEY (`_id`),
  KEY `cuisine_idx` (`$ix_t20_BC26D4DF1273E3F7412529AEE9E95A0CC8475CEB`(20))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```
Example (5): arrays

```sql
MySQL localhost:33060+ docstore SQL  select doc->>'$.grades' from restaurants limit 1

<table>
<thead>
<tr>
<th>date</th>
<th>grade</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014-03-03</td>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>2013-09-11</td>
<td>A</td>
<td>6</td>
</tr>
<tr>
<td>2013-01-24</td>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>2011-11-23</td>
<td>A</td>
<td>9</td>
</tr>
<tr>
<td>2011-03-10</td>
<td>B</td>
<td>14</td>
</tr>
</tbody>
</table>
```
Example (5): arrays

```
MySQL> select doc->>'$.grades' from restaurants limit 1
G
1. row

doc->>'$.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": "B", "score": 14}]
```

```
MySQL> select doc->>'$.grades[0]' from restaurants limit 1
G
1. row

doc->>'$.grades[0]': {
  "date": "2014-03-03T00:00:00Z", "grade": "A", "score": 2}
```
Example (5): arrays

```sql
MySQL> localhost:33060+ docstore SQL> select doc->>'$.grades' from restaurants limit 1

1. row

"$.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}]
```

```sql
MySQL> localhost:33060+ docstore SQL> select doc->>'$.grades[0]' from restaurants limit 1

1. row

"$.grades[0]": {"date": "2014-03-03T00:00:00Z", "grade": "A", "score": 2}
```

```sql
MySQL> localhost:33060+ docstore SQL> select doc->>'$.grades[last]' from restaurants limit 1

1. row

"$.grades[last]": {"date": "2011-03-10T00:00:00Z", "grade": "B", "score": 14}
```
Example (5): arrays

```sql
MySQL> localhost:33060+ docstore SQL

select doc->>'$.grades' from restaurants limit 1

1. row

doc->>'$.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}]
```

```sql
MySQL> localhost:33060+ docstore SQL

select doc->>'$.grades[0]' from restaurants limit 1

1. row

1. row

doc->>'$.grades[0]': {"date": "2014-03-03T00:00:00Z", "grade": "A", "score": 2}
```

```sql
MySQL> localhost:33060+ docstore SQL

select doc->>'$.grades[last]' from restaurants limit 1

1. row

1. row

doc->>'$.grades[last]': {"date": "2011-03-10T00:00:00Z", "grade": "B", "score": 14}
```

```sql
MySQL> localhost:33060+ docstore SQL

select doc->>'$.grades[1 to 2]' from restaurants limit 1

1. row

1. row

doc->>'$.grades[1 to 2]': [{"date": "2013-09-11T00:00:00Z", "grade": "A", "score": 6}, {"date": "2013-01-24T00:00:00Z", "grade": "A", "score": 10}]
```
Example (5): JSON_TABLE

```sql
SELECT rname, style, avg(rating) FROM restaurants,
    JSON_TABLE(doc, "$" columns(  
        rname varchar(100) path ".name",
        style varchar(100) path ".cuisine",
        nested path ".grades[*]"
    columns (rating int path ".score")))
AS jt GROUP BY rname, style LIMIT 10;
```

<table>
<thead>
<tr>
<th>rname</th>
<th>style</th>
<th>avg(rating)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Morris Park Bake Shop</td>
<td>Bakery</td>
<td>8.2000</td>
</tr>
<tr>
<td>Wendy'S</td>
<td>Hamburgers</td>
<td>9.4404</td>
</tr>
<tr>
<td>Dj Reynolds Pub And Restaurant</td>
<td>Irish</td>
<td>9.2500</td>
</tr>
<tr>
<td>Riviera Caterer</td>
<td>American</td>
<td>9.0000</td>
</tr>
<tr>
<td>Tov Kosher Kitchen</td>
<td>Jewish/Kosher</td>
<td>17.7500</td>
</tr>
<tr>
<td>Brunos On The Boulevard</td>
<td>American</td>
<td>17.0000</td>
</tr>
<tr>
<td>Kosher Island</td>
<td>Jewish/Kosher</td>
<td>10.5000</td>
</tr>
<tr>
<td>Wilken'S Fine Food</td>
<td>Delicatessen</td>
<td>10.0000</td>
</tr>
<tr>
<td>Regina Caterers</td>
<td>American</td>
<td>9.6000</td>
</tr>
<tr>
<td>Taste The Tropics Ice Cream</td>
<td>Ice Cream, Gelato, Yogurt, Ices</td>
<td>8.2500</td>
</tr>
</tbody>
</table>
what do I gain?

Conclusion
Conclusion

This is the best of the two worlds in one product!

- Data integrity
- ACID Compliant
- Transactions
- SQL

- schemaless
- flexible data structure
- easy to start (CRUD)
They say that ______ is the best thing since sliced bread!

MySQL 8.0 Document Store

Cards AgainstMySQL Community
Thank you!

Any Questions?

share your 💕 for MySQL on social media using