Scott Stroz – MySQL Developer Advocate

MySQL, JSON, & You: Perfect Together

Obligatory “I Love Me” Slide

- Developer for 20+ years
  - Only constant in that stack has been MySQL
- MySQL Developer Advocate for Oracle
- Avid golfer
- Die hard NY Giants fan
- I have the best office mate

What will we cover?

- What is JSON?
- JSON as a string vs. JSON data type
- Why store it in database?
- How to persist JSON data
- How to retrieve JSON data
- Updating JSON data
- Using relational data as JSON
  - And vice versa
- MySQL document store

What is JSON?

- JSON – *J*ava*S*cript *O*bject *N*otation
- Textual representation of a data structure
  - Objects are wrapped in { }
    - Properties are key value pairs
      - Keys are wrapped in " "
  - Arrays are wrapped in [ ]
  - Data can be nested.
    - Objects can have properties that are arrays of objects.
  - Language independent

Normalize until it hurts; denormalize until it works.

Unknown

JSON Syntax


{
"firstName" : "Scott",
"lastName" : "Stroz",
"age" : 53,
"isOld" : true,
"friends" : [
{
"firsName" : "Ray",
"lastName" : "Camden",
"age" : 46,
"isOld" : false
},
{
"firsName" : "Todd",
"lastName" : "Sharp",
"age" : 47,
"isOld" : false
}
]
}

JOSN as String vs. JSON Data type

JSON as a String

- **Was used long before JSON data type**
  - **Stored as `CHAR`, `VARCHAR`, `TEXT`, etc**
- **Searching by values required the use of `LIKE` or `REGEXP`**
- **Updating any value of the JSON object would require rewriting the entire string**

JSON Data type

- **Introduced in MySQL 5.7**
- **Designed to hold *valid* JSON documents.**
- **Stored in a binary format**
  - **Optimized for quick searches**
- **Can have a defined schema**

Why store JSON?

- Faster development time
- Less verbose than XML
- Pretty much every programming language can ‘read’ JSON
- Some data simply does not need to be normalized
  - Examples:
    - User preferences
    - Configuration data
    - Feature flags

Image by Tumisu from Pixabay

DISCLAIMER

Persisting JSON

- First, we need a table with a JSON column

```
CREATE TABLE season (
    `id` int NOT NULL,
    `name` VARCHAR(100) NOT NULL,
    `season_settings` JSON,
    `start_date` DATE,
    `league_id` int NOT NULL,
    PRIMARY KEY (`id`)
);
```

```
+-----------------+--------------+------+-----+---------+-------+
| Field           | Type         | Null | Key | Default | Extra |
+-----------------+--------------+------+-----+---------+-------+
| id              | int          | NO   | PRI | NULL    |       |
| name            | varchar(100) | NO   |     | NULL    |       |
| season_settings | json         | YES  |     | NULL    |       |
| start_date      | date         | YES  |     | NULL    |       |
| league_id       | int          | NO   |     | NULL    |       |
+-----------------+--------------+------+-----+---------+-------+
```

Insert JSON using... INSERT

```
INSERT INTO season (name,
    start_date,
    league_id,
    season_settings
)
VALUES( 'My Test League',
    now(),
    1,
    '{}'
);
```

```
+----+-----------+----------------+------------+-----------------+
| id | league_id | name           | start_date | season_settings |
+----+-----------+----------------+------------+-----------------+
| 24 |         1 | My Test League | 2022-07-13 | {}              |
+----+-----------+----------------+------------+-----------------+
```

Pathing

- Many of the JSON functions can use a 'path' to nested data
  - $ is considered the 'root'
- JSON_KEYS() will show you the names of the keys at the current level

```
SELECT
JSON_KEYS(season_settings)
FROM season
WHERE id = 23;
```

```
["course", "scoring", "subPool",
"useSubs", "leagueFee", "greensFees",
"useContests", "pointsPerHole",
"golfersPerTeam", "pointsPerMatch",
"maximumHandicap", "maximumTeamSubs",
"pointsForForfeit",
"maximumGolferSubs",
"maximumScorePerHole",
"blindMatchForForfeit",
"maximumForfeitPoints",
"maximumScoresFOrHandicap"]
```

Pathing (continued)

- If we want to get the keys for the 'scoring' property we use the following:

```
SELECT
JSON_PRETTY(
JSON_KEYS(season_settings,
"$.scoring")
)
FROM season
WHERE id = 23;
```

```
[
  "title",
  "description",
  "scoringFile",
  "handicapType",
  "scorePerGolfer"
]
```

Querying based on JSON values

Photo by Caspar Camille Rubin on Unsplash

JSON_CONTAINS()

- Checks if the value of a key equals a specific value.

```
SELECT `id`, `name`
FROM season
WHERE
JSON_CONTAINS(season_settings,
"70", "$.Leaguefee");
```

id	name
8	Summer 2016
11	Summer 2017
14	Summer 2018
16	Summer 2019
18	Summer 2020
21	Summer 2021
23	Summer 2022

Using JSON_CONTAINS() with strings

- When using JSON_CONTAINS() to search for a string, we need to double quote the value.

```
SELECT `id`, `name`
FROM season
WHERE
JSON_CONTAINS(season_settings,
'"Charles Town"',
"$.course.city");
```

id	name
2	Summer 2013
4	Summer 2014
6	Summer 2015
8	Summer 2016
11	Summer 2017
14	Summer 2018
16	Summer 2019
18	Summer 2020
21	Summer 2021
23	Summer 2022

JSON_VALUE()

id	name	greens_fees
2	Summer 2013	15.00
4	Summer 2014	17.50
6	Summer 2015	17.50
8	Summer 2016	17.50
11	Summer 2017	19.50
14	Summer 2018	19.50
16	Summer 2019	19.50
18	Summer 2020	19.50
21	Summer 2021	19.50
23	Summer 2022	19.50

- Can return not only the value, but can be CAST to different data types.

```
SELECT `id`, `name`,
JSON_VALUE(season_settings,
"$.greensFees" RETURNING
DECIMAL(4,2)) AS greens_fees
FROM season
WHERE
JSON_VALUE(season_settings,
"$.course.state" ) = 'WV';
```

Updating JSON values

Photo by Caspar Camille Rubin on Unsplash

How do we update keys/values?

`JSON_INSERT()`

- Inserts a new key to a JSON document
- Will NOT update value for existing keys
- Can add multiple keys in a single statement

`JSON_REPLACE()`

- Updates values to existing keys in a JSON document
- Will add the key if it does not exist
- Can update multiple keys in a single statement

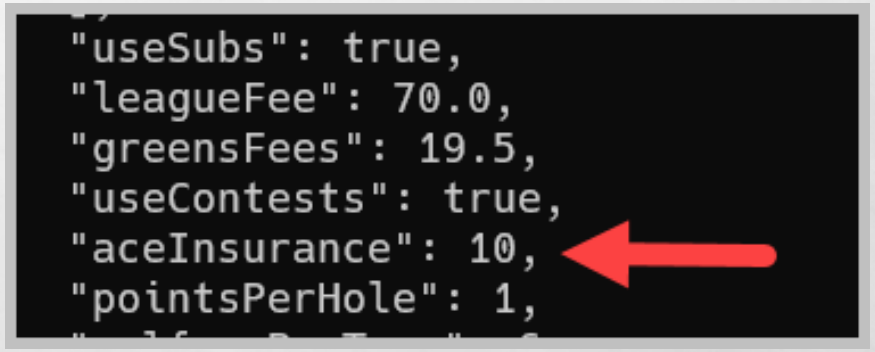
`JSON_SET()`

- Inserts to updates values in a JSON document.
- If the key exists, the old value is updated.
- If the key does not exist, it is added and the new value is used

JSON_INSERT()

```
UPDATE season
SET season_settings =
JSON_INSERT(season_settings,
"$.aceInsurance", 10 );
```

```
+--------------------------+
| settings                 |
+--------------------------+
| {
  "aceInsurance": 10
} |
+--------------------------+
```

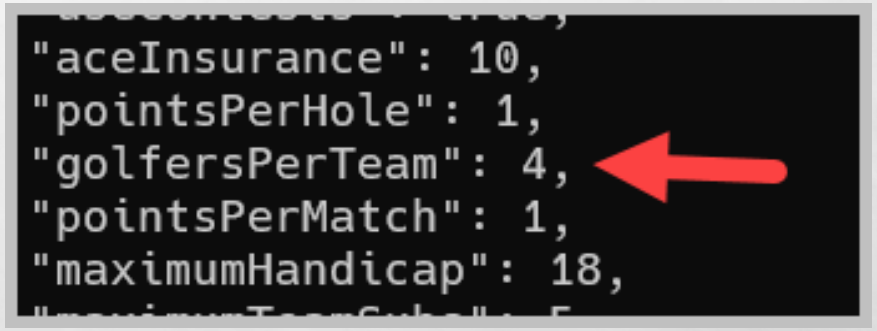


The entire schema is too big to show on one slide.

JSON_REPLACE()

```
UPDATE season SET
season_settings =
JSON_REPLACE(season_settings,
"$.golfersPerTeam", 4 );
```

```
+--------------------------+
| settings                 |
+--------------------------+
| {
  "aceInsurance": 10
} |
+--------------------------+
```



The entire schema is too big to show on one slide.

JSON_SET()

```
UPDATE season SET
season_settings =
JSON_SET(season_settings,
"$.rules", JSON_ARRAY() );
```

```
+----------------------
| settings
+----------------------
| {
  "rules": [],
  "aceInsurance": 10
} |
+----------------------
```

The entire schema is too big to show on one slide.

JSON_REMOVE()

- Used to remove keys form a JSON document.
  - Can remove multiple in a single command.

```
UPDATE season SET
season_settings =
JSON_REMOVE(season_settings,
"$.aceInsurance" );
```

```
+--------------------------+
| settings                 |
+--------------------------+
| {
  "golfersPerTeam": 6
} |
+--------------------------+
```

```
 "leagueFee": 70.0,
 "greensFees": 19.5,
 "useContests": true,
 "pointsPerHole": 1,
 "golfersPerTeam": 6,
 "pointsPerMatch": 1,
 "maximumHandicap": 18,
```

The entire schema is too big to show on one slide.

Using Relational Data as JSON (and vice versa)

Photo by Caspar Camille Rubin on Unsplash

Using JSON_OBJECT()

```
SELECT JSON_PRETTY(
JSON_OBJECT( 'id', id,
'name', name,
'leagueId', league_id,
'startDate', start_date,
'seasonSettings', season_settings)
)
from season where id = 24;
```

```
+-----------------------------------
| season
+-----------------------------------
| {
  "id": 24,
  "name": "My Test League",
  "leagueId": 1,
  "startDate": "2022-07-13",
  "seasonSettings": {
    "golfersPerTeam": 6
  }
} |
+-----------------------------------
```

Using JSON_ARRAY()

```
SELECT JSON_PRETTY(
JSON_ARRAY(
JSON_OBJECT( 'id', id,
'name', name,
'leagueId', league_id,
'startDate', start_date)
)
) seasons
FROM season
WHERE id in (23,24)
ORDER BY start_date DESC;
```

```
+-----------------------------------------
| seasons
+-----------------------------------------
| [
  {
    "id": 24,
    "name": "My Test League",
    "leagueId": 1,
    "startDate": "2022-07-13"
  }
] |
| [
  {
    "id": 23,
    "name": "Summer 2022",
    "leagueId": 1,
    "startDate": "2022-04-12"
  }
]   |
```

Returning JSON Data as Relational Data

```
SELECT name, start_date,
season_settings->>"$.Course.Name" course_name,
CAST(season_settings->>"$.Greensfees" AS DECIMAL(4,2))
greens_fees
FROM season
WHERE year(start_date) >2019
ORDER BY start_date DESC;
```

name	start_date	course_name	greens_fees
My Test League	2022-07-13	NULL	NULL
Summer 2022	2022-04-12	Locust Hill Golf Course	19.50
2021 - Fall Season	2021-08-31	Musket Ridge Golf Club	30.00
2021 - Spring Season	2021-04-12	Glade Valley Golf Club	20.00
Summer 2021	2021-04-12	Locust Hill Golf Course	19.50
2020 - Spring Season	2020-04-16	Glade Valley Golf Club	18.00
Summer 2020	2020-04-14	Locust Hill Golf Course	19.50

MySQL Document Store

Photo by Caspar Camille Rubin on Unsplash

MySQL Document Store

What is it?

- JSON document storage solution built on top of MySQL
- Stored in MySQL table but abstracted from the user
  - ACID compliant
  - InnoDB storage engine
  - Uses JSON datatype
- Simple CRUD API

How can we use it?

- X Plugin
  - Installed by default since 8.0.1
- Uses X Protocol through MySQL Connectors
  - Node.js
  - Java
  - C++
  - Python
  - PHP
  - .Net
  - MySQL Shell

Photo by Fikri Rasyid on Unsplash

Schema for Demo Code


```
{
  "_id": "000062ceaa650000000000000042",
  "date": "2022-06-28",
  "score": 43,
  "course": {
    "par": 36,
    "city": "Charles Town",
    "name": "Locust Hill Golf Course - Back Nine",
    "slope": 132,
    "state": "WV",
    "rating": 36
  },
  "lastName": "Stroz",
  "firstName": "Scott",
  "adjustedScore": 43
}
```

Connecting to the Server

```
\C user:password@localhost
```

```
Creating a session to 'root@localhost'
Fetching schema names for autocompletion... Press ^C to stop.
Your MySQL connection id is 42 (X protocol)
Server version: 8.0.29 MySQL community server - GPL
No default schema selected; type \use <schema> to set one
```

Working with Schemas

- Create a schema

```
session.createSchema('example')
<Schema:example>
```

- Use the schema

```
\use example
default schema `example` accessible through db.
```

- Drop schema

```
session.dropSchema('example')
```

Working with Collections

- Create collection

```
db.Createcollection('round')
```

Result: *<collection:round>*

- Drop collection

```
db.Createcollection('round')
```

Photo by Alvaro Reyes on Unsplash

What a Collection Looks Like in the Database

```
+---------------+----------------+------+-----+---------+-------------------+
| Field         | Type           | Null | Key | Default | Extra             |
+---------------+----------------+------+-----+---------+-------------------+
| doc           | json           | YES  |     | NULL    |                   |
| _id           | varbinary(32)  | NO   | PRI | NULL    | STORED GENERATED  |
| _json_schema  | json           | YES  |     | NULL    | VIRTUAL GENERATED |
+---------------+----------------+------+-----+---------+-------------------+
```

Creating Documents

- Add a single document

```
db.round.add({
"fisrtName": "Scott",
"lastName" : "Stroz",
"date" : "2022-07-12",
"score": 44,
"adjustedScore" : 44,
"course": {"par": 36, "city": "Charles Town",
"name": "Locust Hill Golf Course - Back Nine",
"slope": 132.00, "state": "WV", "rating": 36.00}
})
```

Creating Documents

- Add multiple documents

```
db.round.add([
{
"fisrtName": "Scott", "lastName" : "Stroz",
"date" : "2022-07-12", "score": 44,
"adjustedScore" : 44, "course" :{}
},
{
"fisrtName": "Scott", "lastName" : "Stroz",
"date" : "2022-07-14", "score": 49,
"adjustedScore" : 48, "course" : {}
}
])
```

Searching Documents

- Return all documents

```
db.round.find()
```

- Return documents with search criteria

```
db.round.find("lastName = 'Stroz'")
db.round.find("score < 36")
db.round.find("course.city = :city").bind("city",
"Charles Town")
```

Searching Documents

- Sort results

```
db.round.find().sort(["score"])
```

- Limit results

```
db.round.find().sort(["score"]).limit(1)
```

- Offset results

```
db.round.find().sort(["score"]).limit(5).offset(5)
```

What is returned?


```
{
    "_id": "000062ceaa650000000000002185",
    "date": "2018-04-10",
    "score": 74,
    "course": {
        "par": 36,
        "city": "Frederick",
        "name": "Clustered Spires Golf Club - Front Nine",
        "slope": 128,
        "state": "MD",
        "rating": 34.6
    },
    "lastName": "Thornton",
    "firstName": "Laura",
    "adjustedScore": 74
}
```

Searching Documents

- Return only specific fields

```
db.round.find("score < 36")
.fields(["firstName",
"lastName",
"date",
"score",
"course.name as courseName"])
```

What is returned?


```
{
    "date": "2022-06-20",
    "score": 29,
    "lastName": "Weis",
    "firstName": "Christopher",
    "courseName": "Locust Hill Golf Course - Front Nine"
}
```

Searching Documents

- Grouping results

```
db.round.find()
.fields(['course.name as courseName',
'format(avg(score), 2) * 1  as avg',
'min(score) * 1 as lowestScore',
'max(score) * 1 as highestScore'])
.groupBy(['course.name'])
.sort('course.name')
```

What is returned?


```
{
    "avg": 47.19,
    "courseName": "Locust Hill Golf Course - Back Nine",
    "lowestScore": 32,
    "highestScore": 70
}
{
    "avg": 46.18,
    "courseName": "Locust Hill Golf Course - Front Nine",
    "lowestScore": 29,
    "highestScore": 79
}
```

Using SQL with Document Store

```
with rounds as (
select doc->> '$.firstName' as firstName,
    doc->> '$.lastName' as lastName,
    doc->> '$.score' * 1 as score,
    doc->> '$.course.name' as courseName,
    doc->> '$.date' as datePlayed
from round ),
roundsAgg as ( select courseName, min( score ) lowScore from rounds group by courseName )
select  JSON_PRETTY(JSON_OBJECT(
    'courseName', ra.courseName,
    'score', ra.lowScore,
    'golfers', ( select JSON_ARRAYAGG(JSON_OBJECT('golfer', concat(r.firstName, ' ', r.lastName), 'datePlayed', r.datePlayed ) )
        from rounds r
        where r.score = ra.lowScore and r.courseName = ra.courseName )
    ) ) as data
from roundsAgg ra
group by ra.courseName
order by ra.courseName;
```

What is returned?


```
| {
  "score": 33.0,
  "golfers": [
    {
      "golfer": "Chuck Ripple",
      "datePlayed": "2021-10-19"
    },
    {
      "golfer": "Christian Springsteen",
      "datePlayed": "2021-10-19"
    },
    {
      "golfer": "Chuck Ripple",
      "datePlayed": "2021-09-21"
    },
    {
      "golfer": "Chuck Ripple",
      "datePlayed": "2021-09-07"
    }
  ],
  "courseName": "Musket Ridge Golf Club - Back Nine"
} |
```

RECAP

- WE DEFINED JSON AND TALKED ABOUT SYNTAX.
- JSON AS A STRING VS. JSON DATA TYPE
- WHY STORE IT IN DATABASE?
- HOW TO PERSIST JSON DATA
- HOW TO RETRIEVE JSON DATA
- UPDATING JSON DATA
- USING RELATIONAL DATA AS JSON
- USING JSON AS RELATIONAL DATA
- MYSQL DOCUMENT STORE

Resources

- GitHub
  - https://github.com/boyzoid/mysql-json-demo
- MySQL Documentation
  - https://dev.mysql.com/doc/refman/8.0/en/document-store-concepts.html
  - https://dev.mysql.com/doc/refman/8.0/en/json-function-reference.html
- David Stokes' book
  - MySQL & JSON: A Practical Programming Guide