

# MySQL, JSON, & You: Perfect Together

Scott Stroz – MySQL Developer Advocate



# 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 – JavaScript Object Notation**
- **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**

# Why store JSON?

- Faster development time
  - If schema will change often, it might be better to have no schema
- Less verbose than XML
- Pretty much every programming language can ‘read’ JSON
- Some data is unstructured
  - User preferences
  - Configuration data
  - Feature flags



Image by [Tumisu](#) from [Pixabay](#)

# JSON syntax

```
{  
  "firstName": "Scott",  
  "lastName": "Stroz",  
  "numberOfKids": 2,  
  "playsGolf": true,  
  "friends" : [  
    {  
      "firstName": "Ray",  
      "lastName": "Camden",  
      "numberOfKids": 8,  
      "playsGolf": false,  
    },  
    {  
      "firstName": "Todd",  
      "lastName": "Sharp",  
      "numberOfKids": 2,  
      "playsGolf": true,  
    }  
  ]  
}
```

# 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 replication & quick searches
- Can have a defined schema

# Creating a Table with JSON column

```
CREATE TABLE `season` (
    `id` int NOT NULL AUTO_INCREMENT,
    `league_id` int NOT NULL,
    `name` varchar(100) NOT NULL,
    `start_date` date DEFAULT NULL,
    `season_settings` json DEFAULT NULL,
    CHECK(
        JSON_SCHEMA_VALID(
            {
                "type": "object",
                "properties":{
                    "leagueFees": {
                        "type": "number", "minimum":0
                    }
                }
            },
            `season_settings`
        ),
        PRIMARY KEY (`id`)
);
```



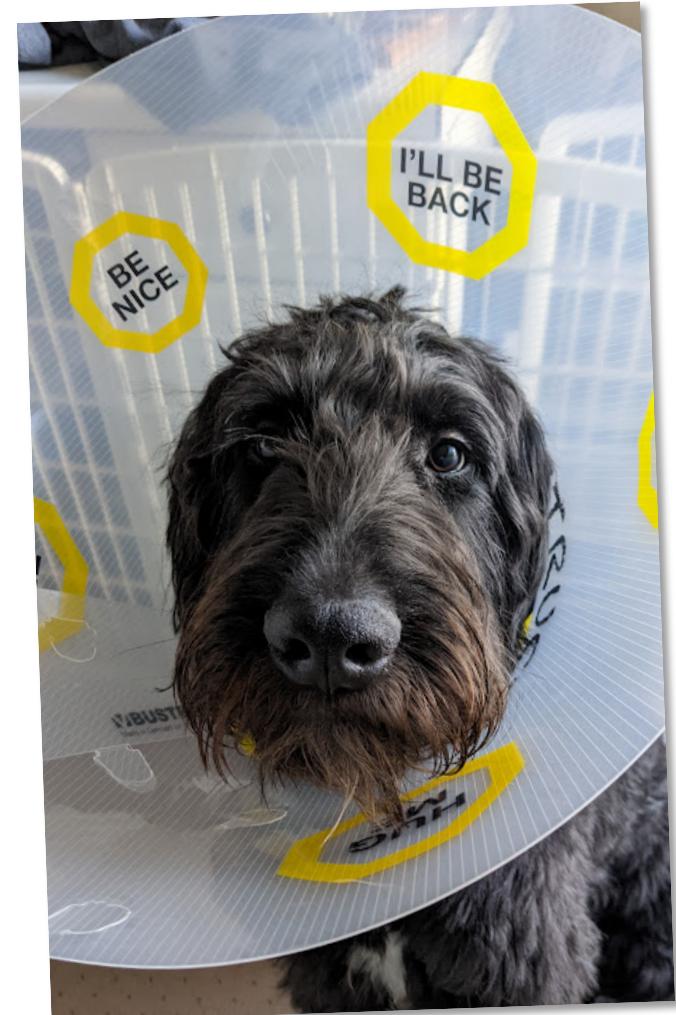
# Insert JSON using... **INSERT**

```
INSERT INTO season (
    name,
    start_date,
    league_id,
    season_settings
)
VALUES( 'My Test League',
        '2022-07-18',
        1,
        '{}'
);
```

+-----+	+-----+	+-----+	+-----+	+-----+
id	league_id	name	start_date	season_settings
+---+---+---+---+---+	+---+---+---+---+---+	+---+---+---+---+---+	+---+---+---+---+---+	+---+---+---+---+---+
24	1	My Test League	2022-07-18	{}
+---+---+---+---+---+	+---+---+---+---+---+	+---+---+---+---+---+	+---+---+---+---+---+	+---+---+---+---+---+

# Demo Schema

```
{  
  "course": {  
    "city": "Charles Town",  
    "name": "Locust Hill Golf Course",  
    "phone": "(304) 728-7300",  
    "state": "WV",  
    "address": "278 St. Andrews Dr.",  
    "postalCode": "25414"  
  },  
  "scoring": {  
    "title": "Better Ball",  
    "description": "Better ball format where we adjust the handicaps",  
    "handicapType": "individual"  
  },  
  "subPool": [  
    {"type": "division", "value": "opposing"},  
    {"name": "ROLE.Course_Staff", "type": "role"},  
    {"name": "ROLE.Golfer_Sub", "type": "role"}  
  ],  
  "useSubs": true,  
  "leagueFee": 70.0,  
  "greensFees": 19.5,  
  "useContests": true,  
  "pointsPerHole": 1,  
  "golfersPerTeam": 2  
}
```



# Pathing

- Many of the JSON functions can use a 'path' to nested data

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

```
[  
    "course",  
    "scoring",  
    "subPool",  
    "useSubs",  
    "leagueFee",  
    "greensFees",  
    "useContests",  
    "pointsPerHole",  
    "golfersPerTeam"  
]
```

# 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",  
    "handicapType"  
]
```

# Querying based on JSON values



Photo by [Caspar Camille Rubin](#) on [Unsplash](#)

# JSON\_CONTAINS()

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

+-----+	id   name   league_fee	+-----+
8   Summer 2016   70.0		
11   Summer 2017   70.0		
14   Summer 2018   70.0		
16   Summer 2019   70.0		
18   Summer 2020   70.0		
21   Summer 2021   70.0		
23   Summer 2022   70.0		

```

SELECT `id`,
       `name`,
       JSON_ARRAYAGG( jt1.role) sub_roles ③
FROM season,
     JSON_TABLE(
           season_settings,
           "$.subPool[*]" COLUMNS(
               subType NVARCHAR(20) PATH '$.type',
               role NVARCHAR(20) PATH '$.name'
           )
     ) as jt1
WHERE
    jt1.subType = 'role'
    AND
    JSON_CONTAINS(
        season_settings,
        '"Charles Town"', ①
        "$.course.city"
    )
GROUP BY season.id;

```

# JSON\_CONTAINS() with strings

id	name	sub_roles
2	Summer 2013	[ "ROLE_GOLFER_SUB" ]
4	Summer 2014	[ "ROLE_GOLFER_SUB" ]
6	Summer 2015	[ "ROLE_GOLFER_SUB" ]
8	Summer 2016	[ "ROLE_GOLFER_SUB" ]
11	Summer 2017	[ "ROLE_GOLFER_SUB" ]
14	Summer 2018	[ "ROLE_GOLFER_SUB" ]
18	Summer 2020	[ "ROLE.Course_STAFF" ]
21	Summer 2021	[ "ROLE.Course_STAFF", "ROLE_GOLFER_SUB" ]
23	Summer 2022	[ "ROLE.Course_STAFF", "ROLE_GOLFER_SUB" ]



# JSON\_VALUE()

```
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';
```

+-----+	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	

# Using Path Operators

```
SELECT `id`,  
       `name`,  
       season_settings->"$.course.name"  
             AS course_name,  
       season_settings->>"$.course.city"  
             AS course_city  
FROM season  
order by id desc limit 5;
```

+-----+	id   name	course_name	course_city
+-----+	24   My Test League	NULL	NULL
	23   Summer 2022	"Locust Hill Golf Course"	Charles Town
	22   2021 - Fall Season	"Musket Ridge Golf Club"	Myersville
	21   Summer 2021	"Locust Hill Golf Course"	Charles Town
	20   2021 - Spring Season	"Glade Valley Golf Club"	Walkersville
+-----+			

# 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 NOT add key if it does not exist
- Can update multiple keys in a single statement

## **JSON\_SET()**

- Inserts and 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

```
UPDATE season
SET
season_settings =
JSON_INSERT(
    season_settings,
    "$.leagueFee",
    25.50
);
```

```
{
    "leagueFee": 25.50
}
```

```
{
    "useSubs": true,
    "leagueFee": 70.0,
    "greensFees": 19.5,
    "useContests": true,
    "pointsPerHole": 1,
    "golfersPerTeam": 2
}
```



# JSON\_INSERT()



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

```
{
    "leagueFee": 25.50
}
```

```
{
    "useSubs": true,
    "leagueFee": 70.0,
    "greensFees": 19.5,
    "useContests": true,
    "pointsPerHole": 1,
    "golfersPerTeam": 4
}
```



# JSON\_REPLACE()



```
UPDATE season
SET
season_settings =
JSON_SET(
    season_settings,
    "$.golfersPerTeam",
    2
);
```

```
{
    "leagueFee": 25.50,
    "golfersPerTeam": 2
}
```

```
{
    "useSubs": true,
    "leagueFee": 70.0,
    "greensFees": 19.5,
    "useContests": true,
    "pointsPerHole": 1,
    "golfersPerTeam": 2
}
```

# JSON\_SET()



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

```
{
    "leagueFee": 25.50
}
```

```
{
    "useSubs": true,
    "leagueFee": 70.0,
    "greensFees": 19.5,
    "useContests": true,
    "pointsPerHole": 1
}
```

# JSON\_REMOVE()



# Using Relational Data as JSON (and vice versa)



Photo by [Caspar Camille Rubin](#) on [Unsplash](#)

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

```
{
    "id": 24,
    "name": "My Test League",
    "leagueId": 1,
    "startDate": "2022-07-22",
    "seasonSettings": {
        "leagueFee": 25.50
    }
}
```

# Using JSON\_OBJECT()

```
SELECT JSON_PRETTY(  
    JSON_ARRAYAGG(  
        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;
```

```
[  
  {  
    "id": 23,  
    "name": "Summer 2022",  
    "leagueId": 1,  
    "startDate": "2022-04-12"  
  },  
  {  
    "id": 24,  
    "name": "My Test League",  
    "leagueId": 1,  
    "startDate": "2022-07-22"  
  }  
]
```

# Using `JSON_ARRAYAGG()`

# Returning JSON Data as Relational Data

```
SELECT name,  
       season_settings->>"$.course.name" course_name,  
       CAST(  
           season_settings->>("$.greensFees"  
                           AS DECIMAL(4,2)  
           ) greens_fees,  
           season_settings->>"$.scoring.handicapType" hcp_type  
   FROM season  
 WHERE year(start_date) >2019  
 ORDER BY start_date DESC;
```

name	course_name	greens_fees	hcp_type
My Test League	NULL	NULL	NULL
Summer 2022	Locust Hill Golf Course	19.50	individual
2021 - Fall Season	Musket Ridge Golf Club	30.00	team
2021 - Spring Season	Glade Valley Golf Club	20.00	team
Summer 2021	Locust Hill Golf Course	19.50	individual
2020 - Spring Season	Glade Valley Golf Club	18.00	team
Summer 2020	Locust Hill Golf Course	19.50	individual

# 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

# Schema for Demo Code

```
{  
  "_id": "000062ceaa650000000000002a75",  
  "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')
```

Use the Schema

```
\use example
```

Drop the Schema

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

# Working with Collections

Create Collection

```
db.createCollection('round')
```

Drop Collection

```
db.dropCollection('round')
```

Photo by [Alvaro Reyes](#) on [Unsplash](#)

# What the 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

Photo by [Alvaro Reyes](#) on [Unsplash](#)

# Creating Documents

```
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}  
})
```

# Searching Documents

Return All Documents

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

Using 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": "000062ceaa650000000000002a75",  
    "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  
}
```

# Returning Specific Keys

```
db.round.find("score < 30")
.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"  
}
```

# Grouping Results

```
db.round.find()
  .fields([
    'course.name as courseName',
    'ROUND(AVG(score), 2) as avg',
    'CAST(min(score) AS SIGNED) AS lowestScore',
    'CAST(max(score) AS SIGNED) 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,
           CAST(doc->> '$.score' AS SIGNED) AS score,
           doc->> '$.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;
```

1

2

3

4

# What is returned?

```
{  
  "score": 33,  
  "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/json-function-reference.html>
  - <https://dev.mysql.com/doc/refman/8.0/en/document-store-concepts.html>
- David Stokes' book
  - MySQL & JSON: A Practical Programming Guide