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",
 "number0fKids": 2,
 "playsGolf": true,
 "friends" : [
   {
     "firstName": "Ray",
     "lastName": "Camden",
     "number0fKids": 8,
     "playsGolf": false,
   },
    {
     "firstName": "Todd",
     "lastName": "Sharp",
     "number0fKids": 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 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 is unstructured
  - Examples:
    - User preferences
    - Configuration data
    - Feature flags

Image by Tumisu from Pixabay

Creating a Table with 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    | auto_increment |
| league_id       | int         | NO   |     | NULL    |                |
| name            | varchar(100)| NO   |     | NULL    |                |
| start_date      | date        | YES  |     | NULL    |                |
| season_settings | json        | YES  |     | NULL    |                |
+-----------------+-------------+------+-----+---------+----------------+
```

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": "Team aggregate",
    "description": "Winner of the hole is the lowest combined net score",
    "handicapType": "team"
  },
  "subPool": [
    {
      "name": "ROLE_GOLFER_SUB",
      "type": "role"
    }
  ],
  "useSubs": false,
  "leagueFee": 65.0,
  "greensFees": 15.0,
  "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       |
+----+-------------+------------+
```

JSON_CONTAINS() with strings

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

```
+----+-------------+--------------+
| id | name        | course_city  |
+----+-------------+--------------+
|  2 | Summer 2013 | Charles Town |
|  4 | Summer 2014 | Charles Town |
|  6 | Summer 2015 | Charles Town |
|  8 | Summer 2016 | Charles Town |
| 11 | Summer 2017 | Charles Town |
| 14 | Summer 2018 | Charles Town |
| 16 | Summer 2019 | Charles Town |
| 18 | Summer 2020 | Charles Town |
| 21 | Summer 2021 | Charles Town |
| 23 | Summer 2022 | Charles Town |
+----+-------------+--------------+
```

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 |
+----+-------------+-------------+
```

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

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

```json
{
  "aceInsurance": 10
}
```

```json
{
  "useSubs": true,
  "leagueFee": 70.0,
  "greensFees": 19.5,
  "useContests": true,
  "aceInsurance": 10,
  "pointsPerHole": 1,
  "golfersPerTeam": 2
}
```

JSON_INSERT()

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

```
{
 "aceInsurance": 10
 }
```

```
{
 "useSubs": true,
  "leagueFee": 70.0,
  "greensFees": 19.5,
  "useContests": true,
  "aceInsurance": 10,
  "pointsPerHole": 1,
  "golfersPerTeam": 4
 }
```

JSON_REPLACE()

```
UPDATE season
SET
season_settings =
JSON_SET(
    season_settings,
    "$.bonusPoint",
    1
);
```

```
{
  "bonusPoint": 1,
  "aceInsurance": 10
}
```

```
{
  "useSubs": true,
  "leagueFee": 70.0,
  "bonusPoint": 1,
  "greensFees": 19.5,
  "useContests": true,
  "aceInsurance": 10,
  "pointsPerHole": 1,
  "golfersPerTeam": 4
}
```

JSON_SET()

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

```
{
   "bonusPoint": 1
}
```

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

JSON_REMOVE()

Using Relational Data as JSON (and vice versa)

Photo by Caspar Camille Rubin on Unsplash

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

```json
{
  "id": 24,
  "name": "My Test League",
  "leagueId": 1,
  "startDate": "2022-07-18",
  "seasonSettings": {
    "bonusPoint": 1
  }
}
```

Using JSON_OBJECT()

```sql
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;
```

```json
[
  {
    "id": 23,
    "name": "Summer 2022",
    "leagueId": 1,
    "startDate": "2022-04-12"
  },
  {
    "id": 24,
    "name": "My Test League",
    "leagueId": 1,
    "startDate": "2022-07-18"
  }
]
```

Using `JSON_ARRAYAGG()`

```sql
SELECT name,
    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                 | course_name             | greens_fees |
+----------------------+-------------------------+-------------+
| My Test League       | NULL                    |        NULL |
| Summer 2022          | Locust Hill Golf Course |       19.50 |
|  2021 - Fall Season  | Musket Ridge Golf Club  |       30.00 |
| 2021 - Spring Season | Glade Valley Golf Club  |       20.00 |
| Summer 2021          | Locust Hill Golf Course |       19.50 |
| 2020 - Spring Season | Glade Valley Golf Club  |       18.00 |
| Summer 2020          | Locust Hill Golf Course |       19.50 |
+----------------------+-------------------------+-------------+
```

Returning JSON Data as Relational Data

MySQL Document Store

Photo by Caspar Camille Rubin on Unsplash

MySQL Document Store

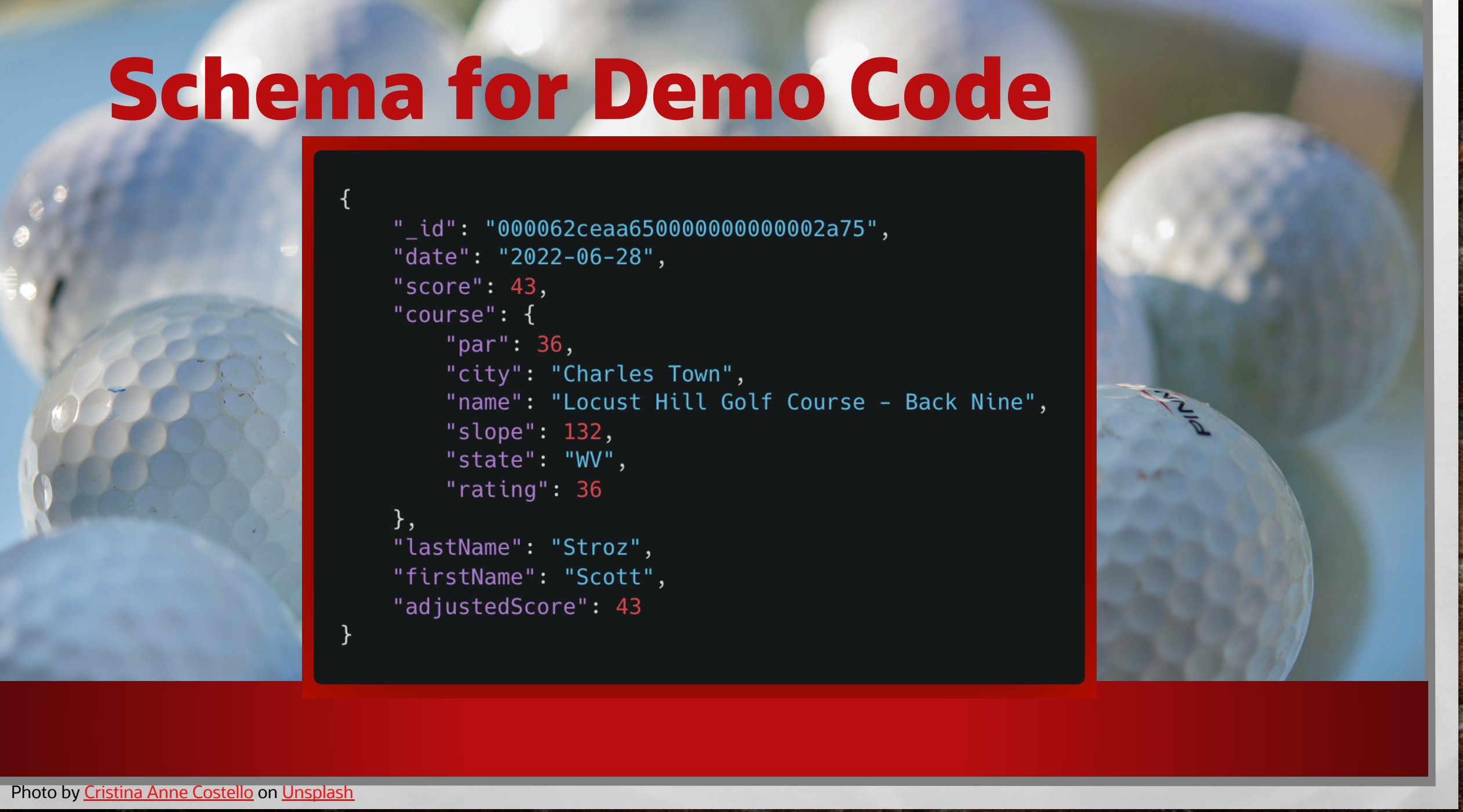
What is it?

- 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')
```

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 |
+--------------+----------------+------+-----+---------+-------------------+
```

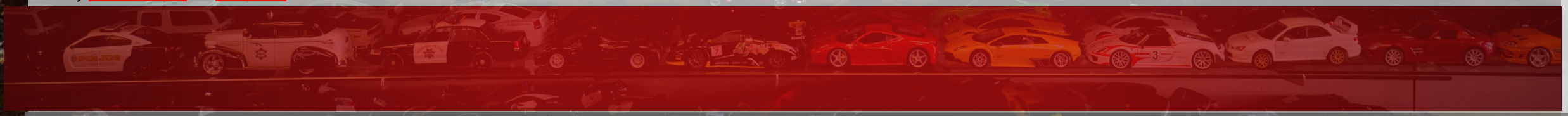


Photo by Alvaro Reyes on Unsplash

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 < 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"
}
```

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/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