ORACLE

MySQL, JSON, & You Perfect Together

Storing and Retrieving JSON Data in MySQL

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?

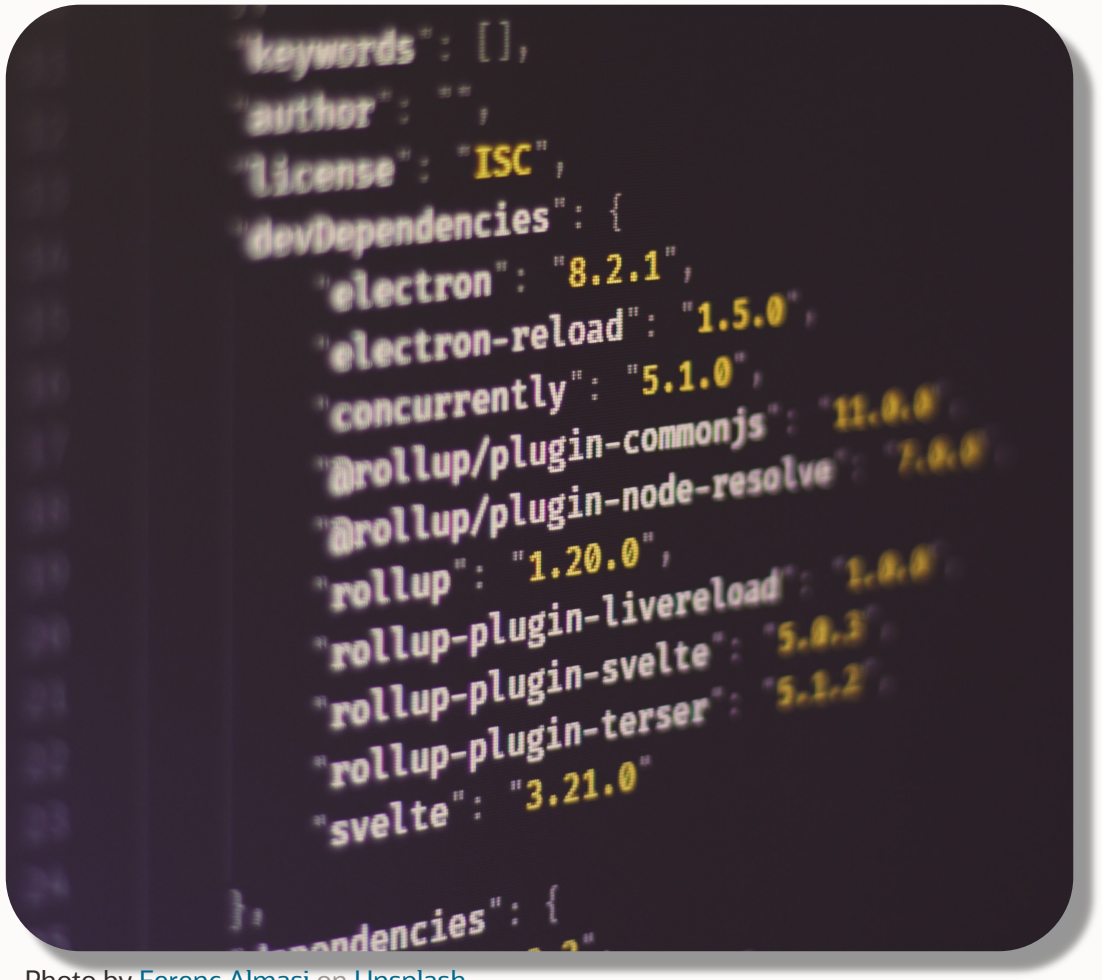


Photo by Ferenc Almasi on Unsplash

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

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

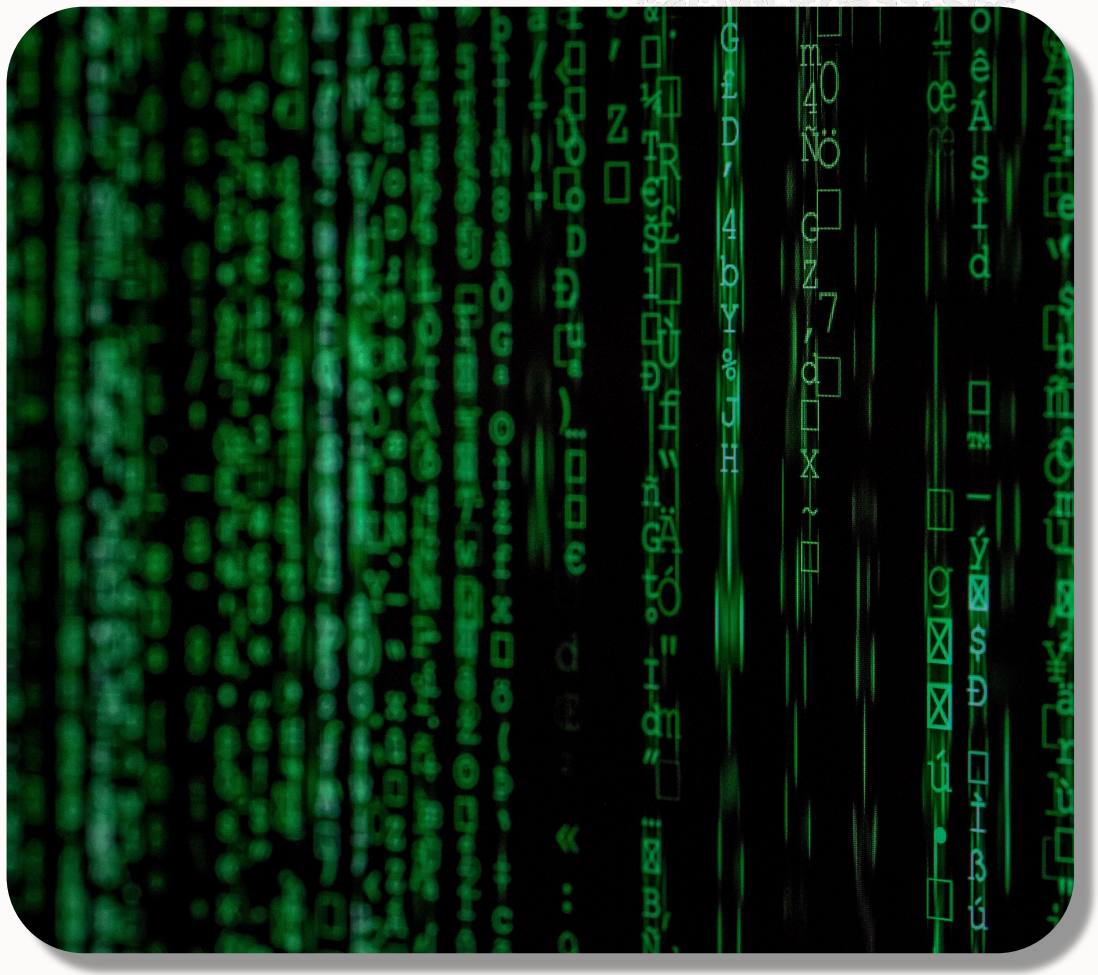


Photo by Markus Spiske on Unsplash

```
{
 "name": "Scott Stroz",
  "numberOfChildren": 2,
  "playesGolf": true,
  "friends": [
    {
      "name": "Raymond Camden",
      "numberOfChildren": 8,
      "playsGolf": false
    },
    {
      "name": "Todd Sharp",
      "numberOfChildren": 2,
      "playsGolf": true
    }
  ]
 }
```

JSON Syntax

"Normalize until it hurts; denormalize until it works."

Unknown

Why Store JSON?

Image by Tumisu from Pixabay

- 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

JSON as String vs. JSON Data Type

JSON as String

- It was used long before the JSON data type existed
  - 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

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
}
```

```
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`)
);
```

Creating a Table with JSON column

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 | {}              |
+----+-----------+----------------+------------+-----------------+
```

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 (con't)

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 data based on JSON values

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_ARRAYAGG( jt1.role) sub_roles  3
FROM season,
   2 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;
```

```
+----+-------------+-----------------------------------------+
| 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"] |
+----+-------------+-----------------------------------------+
```

4

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

How do we update keys/values?

JSON_INSERT() JSON_REPLACE() JSON_SET()

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

- 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

- 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

JSON_INSERT()

```
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_REPLACE()

```
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_SET()

```
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_REMOVE()

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

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

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

Using Relational Data as JSON

And Vice Versa

Using JSON_OBJECT()

```
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_ARRAYAGG()

```
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"
  }
]
```

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;
```

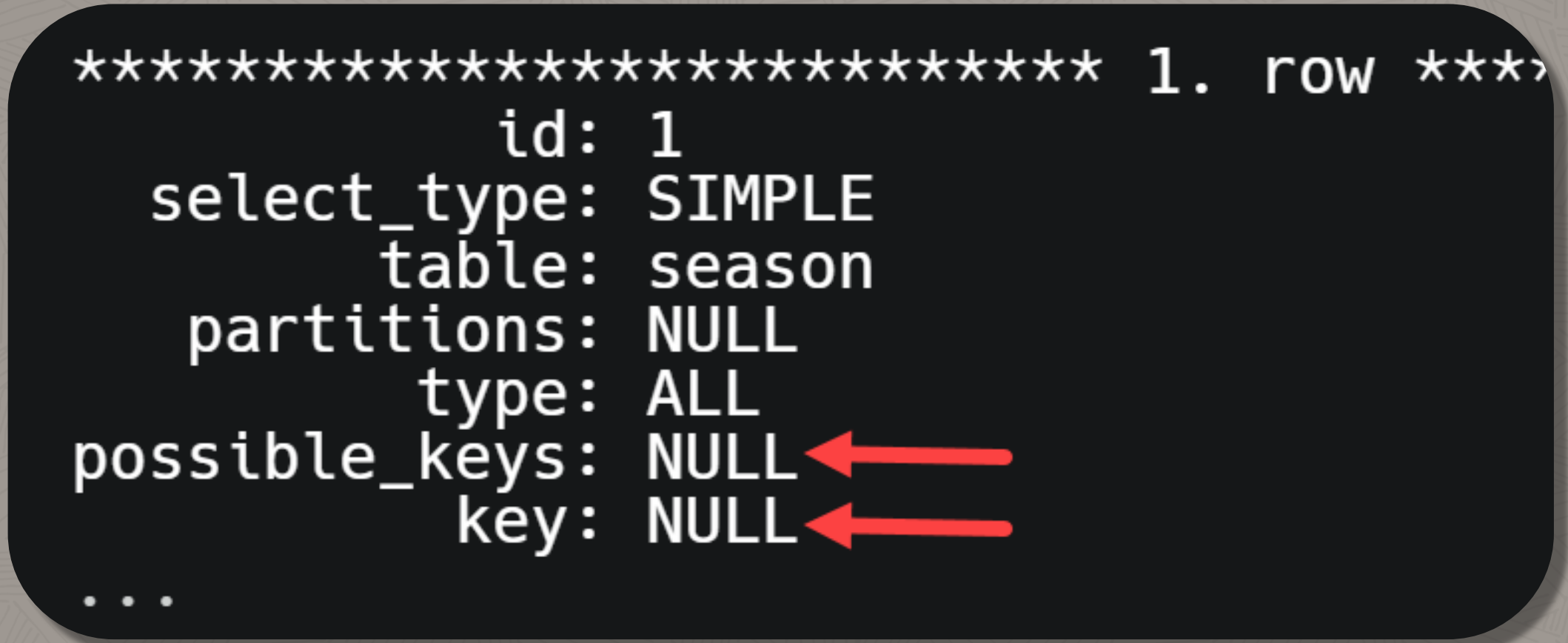
```
+---------------------+-------------------------+-------------+------------+
| name                | course_name             | greens_fees | hcp_type   |
+---------------------+-------------------------+-------------+------------+
| My Test League      | NULL                    |        NULL | NULL       |
| 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 |
+---------------------+-------------------------+-------------+------------+
```

Indexing JSON values

A Basic Query of JSON Data

```
explain
    select *
    from season
    where season_settings->>'$.course.name'
    = 'Locust Hill Golf Course'\G
```


A Basic Query of JSON Data



Adding an Index for JSON Data

```
ALTER TABLE season
ADD INDEX course_name ((
1 CAST(season_settings->>'$.course.name'
        as CHAR(255)) 2
3 COLLATE utf8mb4_bin
));
```

What does the new index look like?

```
show indexes from season\G
```

What does the new index look like?

```
*************************** 2. row ***************************
        Table: season
   Non_unique: 1
     Key_name: course_name
 Seq_in_index: 1
  Column_name: NULL
    Collation: A
  Cardinality: 4
     Sub_part: NULL
       Packed: NULL
         Null: YES
   Index_type: BTREE
      Comment:
Index_comment:
      Visible: YES
   Expression: (cast(json_unquote(json_extract(`season_settings`,_utf8mb4\'$.course.name\'))
as char(255) charset utf8mb4) collate utf8mb4_bin)
```

What Does the Explain Plan Look Like Now?

```
explain
    select *
    from season
    where season_settings->>'$.course.name'
    = 'Locust Hill Golf Course'\G
```

What Does the Explain Plan Look Like Now?

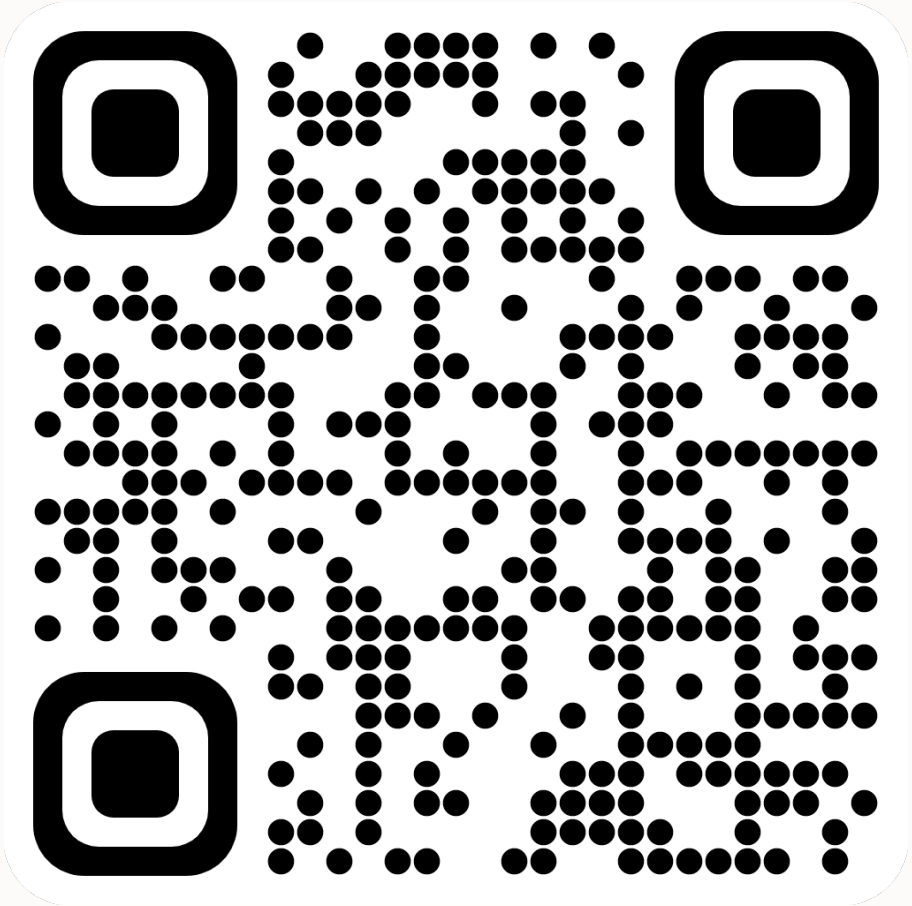
```
*************************** 1. row ***************
           id: 1
  select_type: SIMPLE
        table: season
   partitions: NULL
         type: ref
possible_keys: course_name
          key: course_name
...
```

RECAP

- We defined JSON and talked about syntax.
- JSON as a string vs. JSON data type
- Why store it in a database?
- How to persist JSON data
- How to retrieve JSON data
- Updating JSON data
- Using relational data as JSON
- Using JSON as relational data
- Indexing JSON data

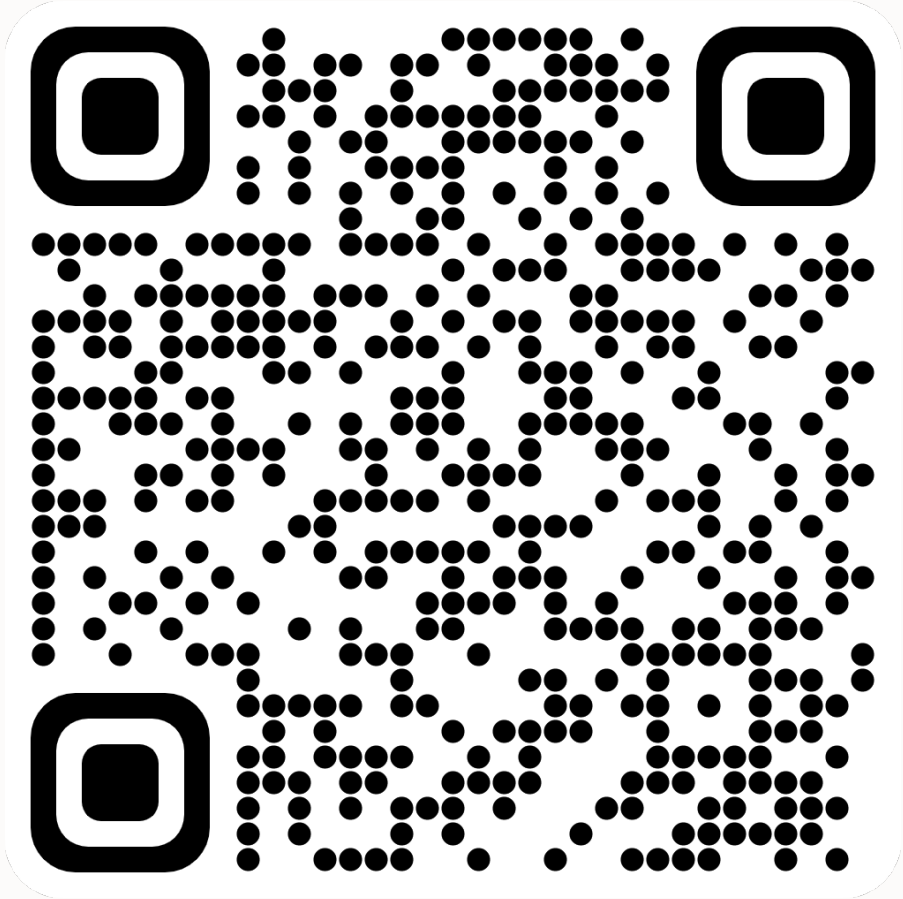
Resources

GitHub Repo



Resources

MySQL JSON Function Documentation



Would you like credit for a MySQL Certification Exam or Training?

See me after the session!

Q&A

Thank You!