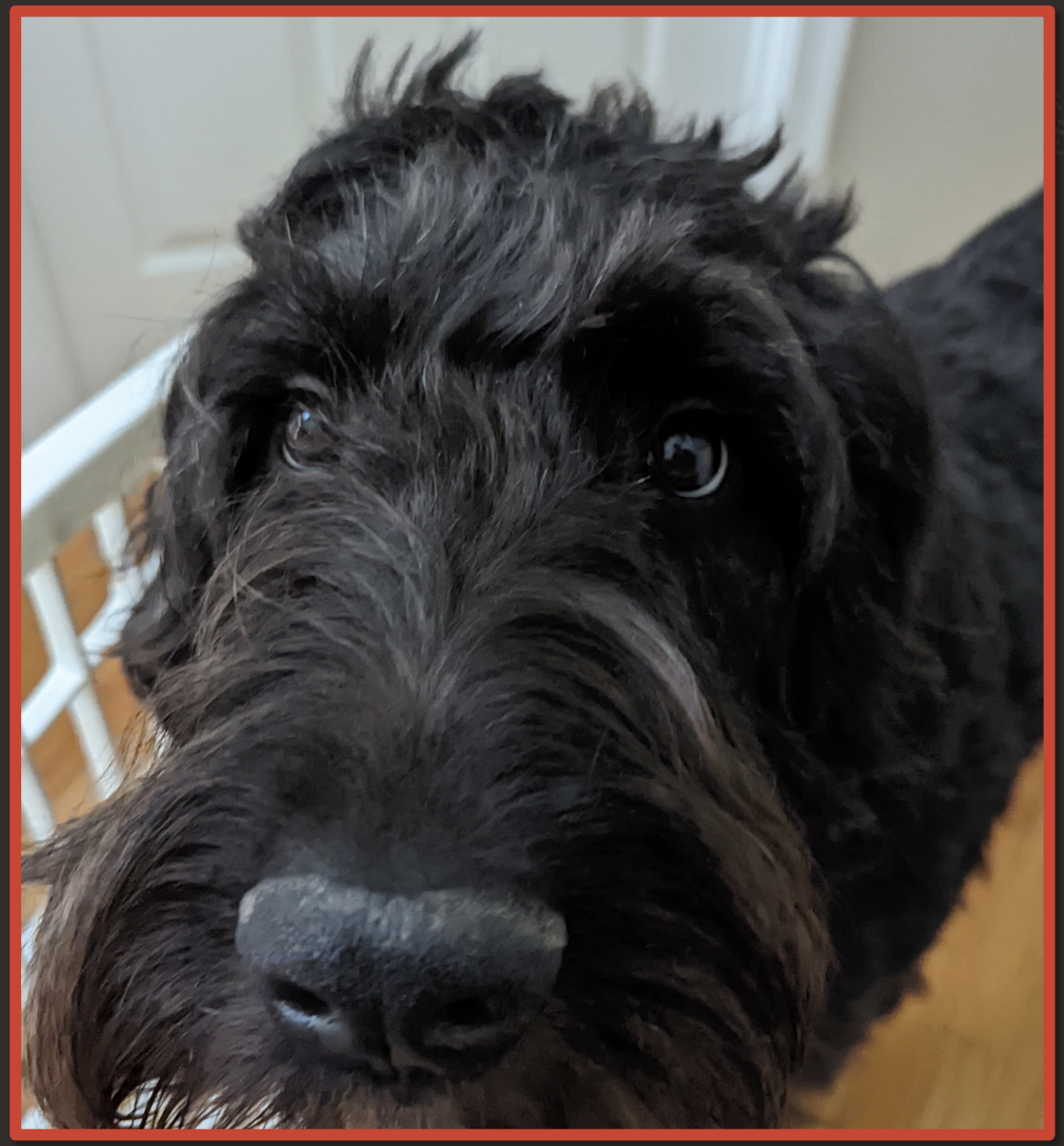
ORACLE

Using MySQL Document Store with Node.js

Scott Stroz
MySQL Developer Advocvate

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 Alexander Grey on Unsplash

- "NoSQL" vs relational databases
- What is MySQL Document Store?
- The anatomy MySQL Document Store data
- How to access MySQL Document Store
- Connecting to MySQL Document Store with Node.js
- Using the CRUD API
- Using raw SQL to query our documents

"NoSQL" vs Relational Data

NoSQL

- Faster development time
- Easier to modify the 'schema'
- Simple CRUD API

- Data is unstructured
- Difficult to run queries for reporting

Relational Data

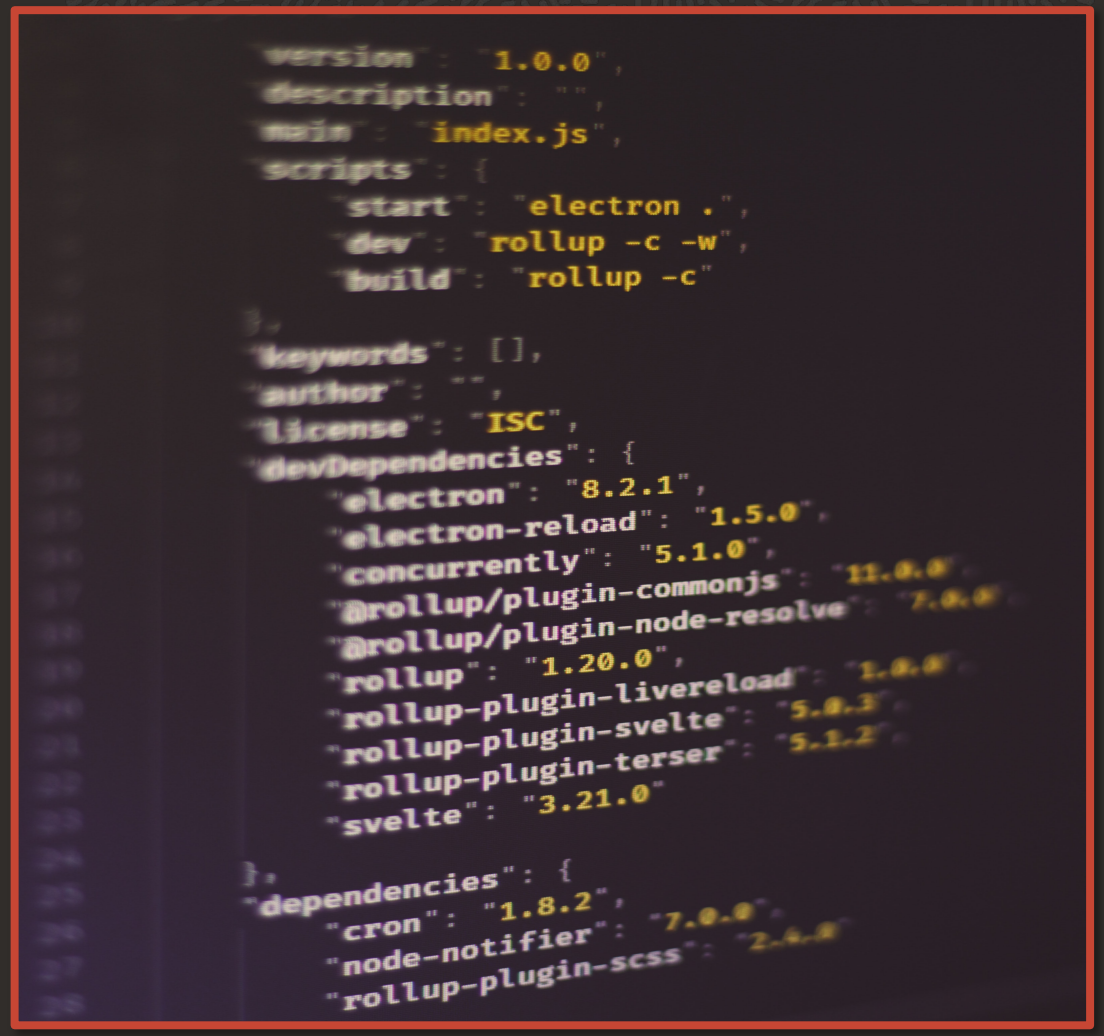
- Data can be structured and organized
- Easier to run queries for reporting

- Slower development time
- As more related data is added, CRUD operations can get more complex

What if I told you you could have the best of both worlds?

What is MySQL Document Store?

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



Anatomy of MySQL Document Store

Photo by Kira auf der Heide on Unsplash

- **Schema**
  - Schema - database
- **Collection**
  - Table
- **Document**
  - Row in table

- **Table Columns**
  - **_id**
    - varbinary(32)
  - **doc**
    - JSON
  - **_json_schema**
    - JSON

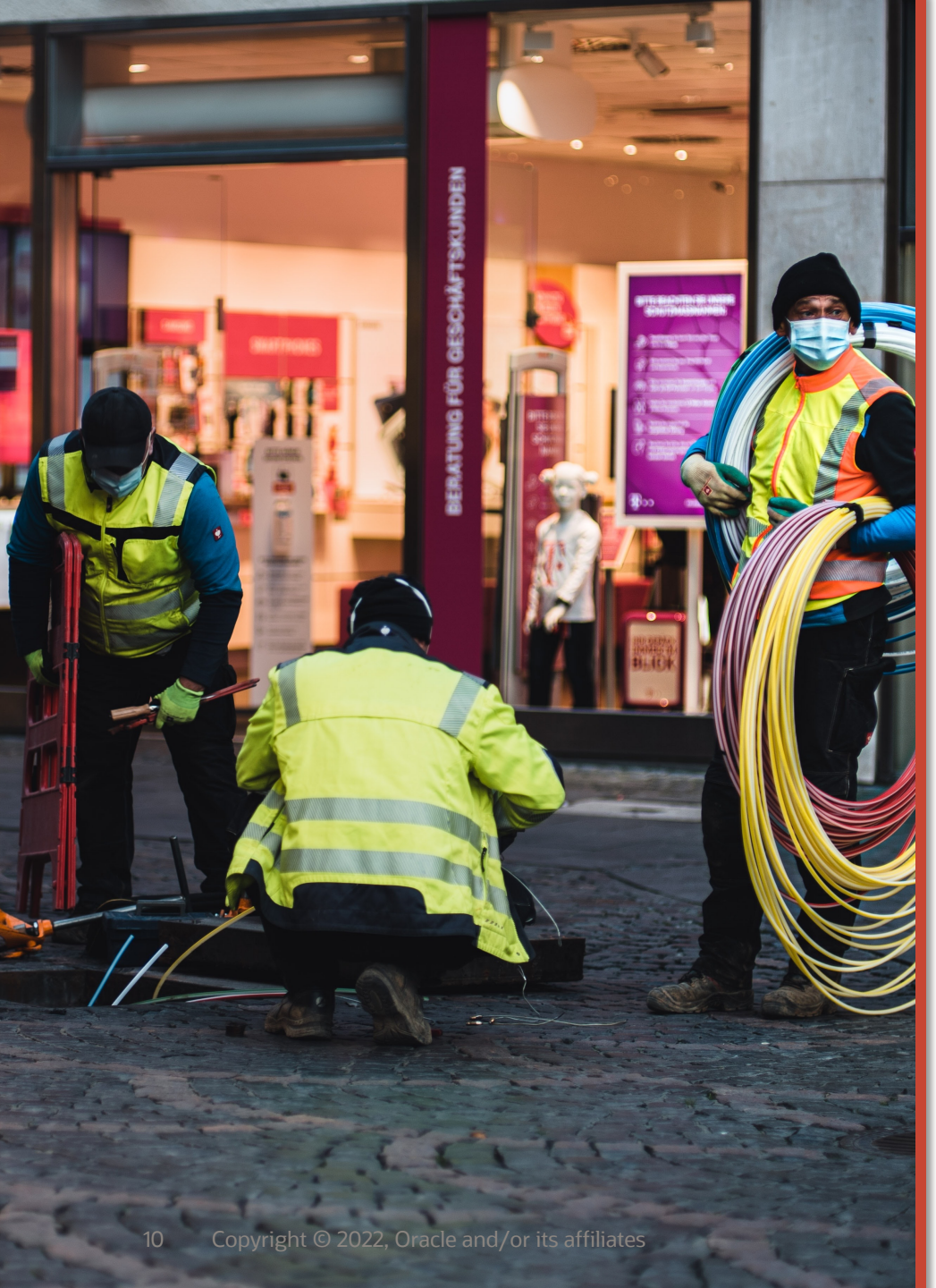
How do we access MySQL Document Store?

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

Photo by Museums Victoria on Unsplash

MySQL Shell

- MySQL Shell is a command line interface for managing MySQL Instances
  - MySQL Document Store schemas and collections
    - Including importing data
  - MySQL Clusters
  - MySQL Replication
  - Run SQL queries



Connecting to MySQL Document Store with Node

Installing the connector

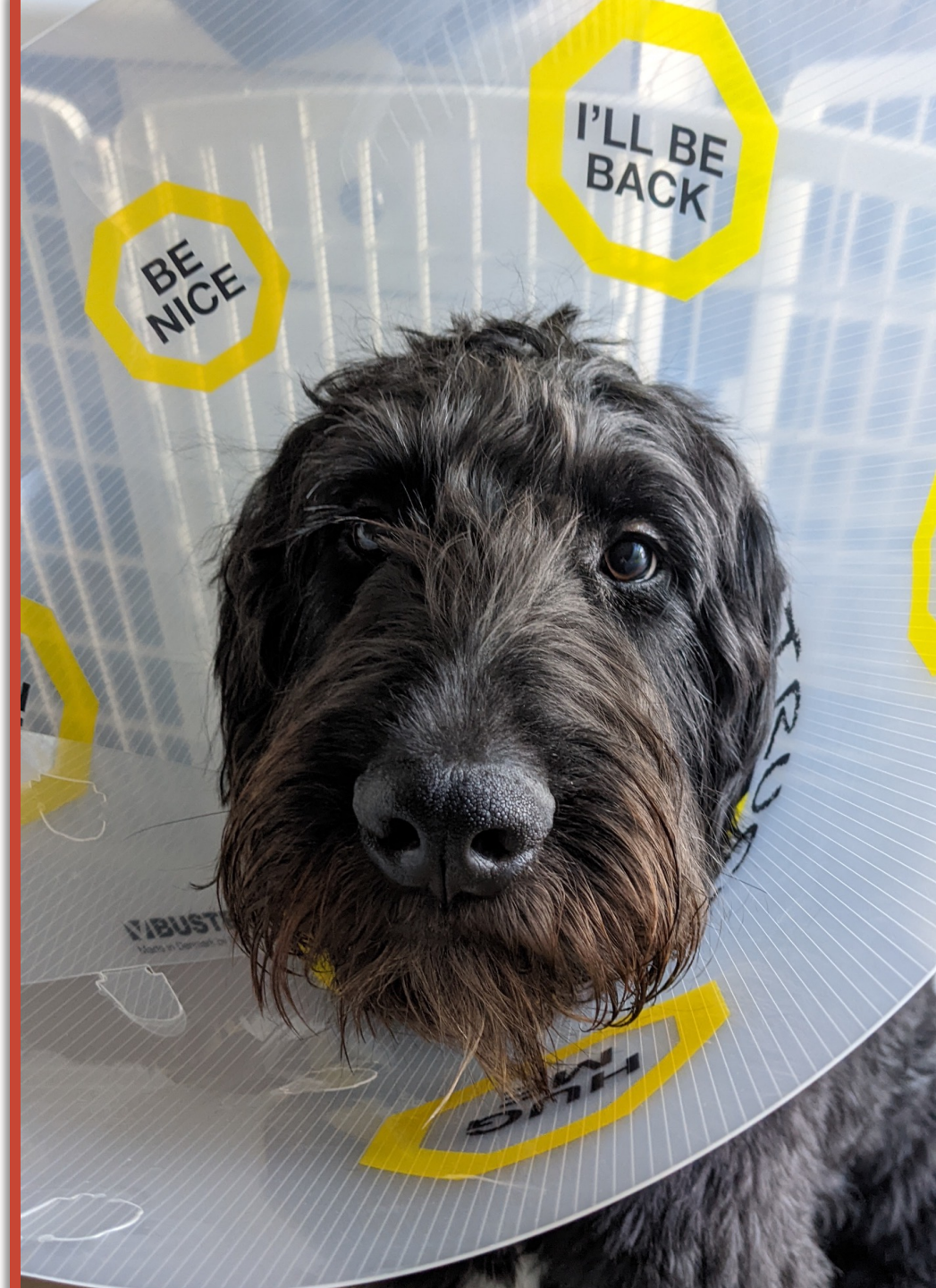
```
npm install @mysql/xdevapi
```


Photo by Mika Baumeister on Unsplash

Connecting to MySQL Document Store with Node

Create Connector Object

```
const mysqlx = require('@mysql/xdevapi')
```



Connecting to MySQL Document Store with Node

Create Connection String

```
const connectionUrl =
  'mysqlx://myUser:myPassword@localhost:33060/node_demo'
```

1 2 3 4 5 6

Photo by Susanne Jutzeler, suju-foto : https://www.pexels.com/photo/brown-thread-2072872/

Connecting to MySQL Document Store with Node

Create Connection Pool

```
const pool = mysqlx.getClient(connectionUrl, {
  pooling: {
    enabled: true,
    maxSize: 10,
    maxIdleTime: 20000,
    queueTimeout: 5000
  }
})
```

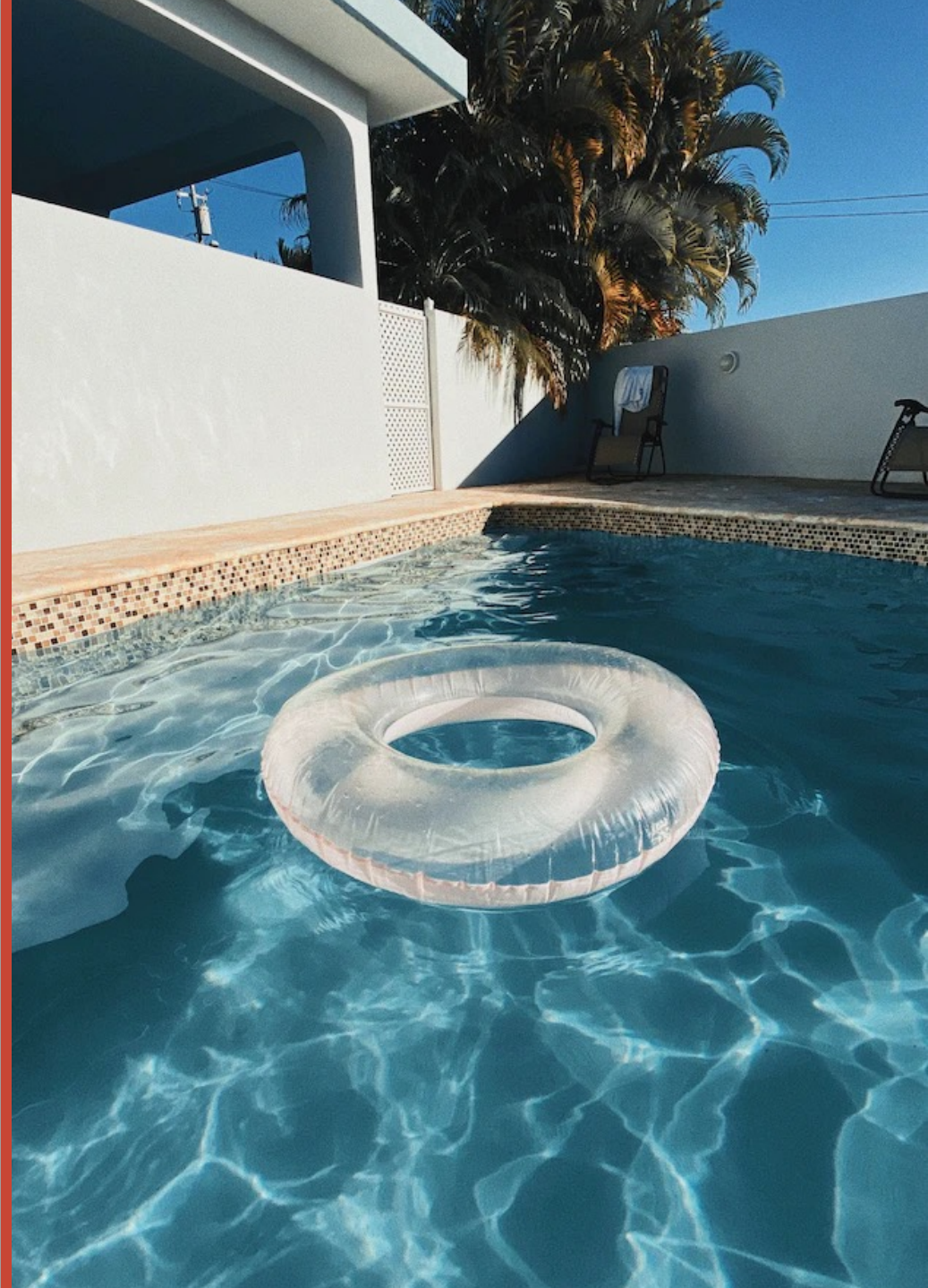


Photo by diego acosta on Unsplash

Schema Used for this Demo


```
{
            "_id": "000062f71e940000000000011129",
            "date": "2022-08-11",
            "score": 45,
            "course": {
                "par": 36,
                "city": "Charles Town",
                "name": "Locust Hill Golf Course - Back Nine",
                "state": "WV"
            },
            "lastName": "Stroz",
            "firstName": "Scott",
            "holeScores": [
                {
                    "par": 4,
                    "score": 4,
                    "number": 11,
                    "relationToPar": 0
                },
                {
                    "par": 4,
                    "score": 5,
                    "number": 14,
                    "relationToPar": 1
                },
                ...
            ]
        }
```

CRUD API

find()

```
// /list endpoint
app.get('/list/', async (req, res ) =>{
    const scores = await listAllScores()
    let msg = { count: scores.length, scores: scores }
    res.send( msg )
```

```
// find() demo
const listAllScores = async () =>{
    let scores = []
    const session = await pool.getSession() 1
    const db = session.getSchema(databaseName) 2
    const collection = db.getCollection(collectionName) 3
    await collection.find() 4
      5 .execute((score) => {
            scores.push(score) 6
        });

    session.close() 7
    return scores
}
```

CRUD API

limit()

```
// /list with limit
app.get('/list/:limit', async (req, res ) =>{
    const scores = await limitAllScores(req.params.limit)
    let msg = { count: scores.length, scores: scores }
    res.send( msg )
```

```
// Showing find() with limit()
const limitAllScores = async ( limit ) =>{
    let scores = []
    const session = await pool.getSession()
    const db = session.getSchema(databaseName)
    const collection = db.getCollection(collectionName)
    await collection.find()
        .limit(limit)
        .execute((score) => {
            scores.push(score)
        });

    session.close()
    return scores
}
```

CRUD API

sort() & fields()

```
// /bestScores endpoint
app.get('/bestScores/:limit?', async (req, res) =>{
  1 let limit = req.params.limit ? req.params.limit : defaultResultLength
    const scores = await getBestScores(limit) 2
    let msg = { count: scores.length, scores: scores }
    res.send( msg )
})
```

```
// showing find() with fields() and sort()
const getBestScores = async ( limit ) =>{
    let scores = []
    const session = await pool.getSession()
    const db = session.getSchema(databaseName)
    const collection = db.getCollection(collectionName)
    await collection.find()
      4 .fields(['firstName', 'lastName', 'score', 'date', 'course.name as courseName']) 5
      3 .sort(['score asc', 'date desc'])
        .limit( limit )
        .execute((score) => {
            scores.push(score)
        });
    session.close()
    return scores
}
```

CRUD API

`find()` with condition

```
// Showing find() with numeric comparison
const getRoundsUnderPar = async () =>{
    let scores = []
    const session = await pool.getSession()
    const db = session.getSchema(databaseName)
    const collection = db.getCollection(collectionName)
    await collection.find("score < course.par") 1
        .fields(['firstName', 'lastName', 'score', 'date', 'course.name as courseName'])
        .execute((score) => {
            scores.push( score )
        })
    session.close()
    return scores
}
```

CRUD API

find() & bind()

```
app.get('/getByScore/:score?', async (req, res) =>{
  1 let score = req.params.score ? req.params.score : 36
    const scores = await getByScore(parseInt(score))2
    let msg = { count: scores.length, scores: scores }
    res.send( msg )
```

```
//show find() with bind()
const getByScore = async ( score ) =>{
    let scores = []
    const session = await pool.getSession()
    const db = session.getSchema(databaseName)
    const collection = db.getCollection(collectionName)
    await collection.find("score = :score") 3
        .bind( 'score', score)4
        .fields(['concat( firstName, " ", lastName) as golfer',
                 5 'score',
                 'date',
                 'course.name as courseName'])
        .sort(['date desc'])
        .execute((score) => {
            scores.push(score)
        });
    session.close()
    return scores
}
```

CRUD API

bind() with LIKE

```
app.get('/getByGolfer/:lastName?', async (req, res) =>{
  1 let str = req.params.lastName || ''
    const scores = await getByGolfer(str) 2
    let msg = { count: scores.length, scores: scores }
    res.send( msg )
})
```

```
const getByGolfer = async ( lastName ) =>{
    let scores = []
    const session = await pool.getSession()
    const db = session.getSchema(databaseName)
    const collection = db.getCollection(collectionName)
    await collection.find("lower(lastName) like :lastName") 3
        .bind('lastName', lastName.toLowerCase()+'%') 4
        .sort(['lastName', 'firstName'])
        .execute((score) => {
            scores.push(score)
        });
    session.close()
    return scores
}
```

CRUD API

groupBy()

```
app.get('/getAverageScorePerGolfer/:year?', async (req, res) =>{ 1
    const scores = await getAverageScorePerGolfer( req.params.year )
    let msg = { count: scores.length, scores: scores }
    res.send( msg )
```

```
const getAverageScorePerGolfer = async ( year ) =>{
    let scores = []
    const session = await pool.getSession()
    const db = session.getSchema(databaseName)
    const collection = db.getCollection(collectionName)
    await collection.find( year ? "year(date) = " + year : 'date is not null') 2
        .fields(['lastName',
                 'firstName',
                 'round(avg(score), 2) as avg', 3
                 'count(score) as numberOfRounds'], 4
        .sort([ 'lastName', 'firstName'])
        .groupBy(['lastName', 'firstName']) 5
        .execute((score) => {
            scores.push( score )
        })
    session.close()
    return scores
}
```

CRUD API

`groupBy()` on a child property

```
const getCourseScoringData = async () =>{
    let scores = []
    const session = await pool.getSession()
    const db = session.getSchema(databaseName)
    const collection = db.getCollection(collectionName)
    await collection.find()
        .fields(['course.name as courseName',
                 'round(avg(score), 2)  as avg', 1
                 'cast(min(score) as unsigned) as lowestScore', 2
                 'cast(max(score) as unsigned) as highestScore', 3
                 'count(score) as numberOfRounds']) 4
        .groupBy(['course.name']) 5
        .sort('course.name desc')
        .execute((score) => {
            scores.push( score )
        })
    session.close()
    return scores
}
```

Using Raw SQL

```
const getHolesInOne = async () =>{
    let aces = []
    const session = await pool.getSession()  1

    const sql = `SELECT JSON_OBJECT(  2
            'firstName', doc ->> '$.firstName',
            'lastName', doc ->> '$.lastName',
            'date', doc ->> '$.date',
            'courseName', doc->> '$.course.name',
            'holes', JSON_ARRAYAGG(
                JSON_OBJECT(
                'holeNumber', holeScores.number,
                              'par', holeScores.par
                        )
                    )
        )
        FROM scores,
      3 JSON_TABLE(
        doc, '$.holeScores[*]'
            COLUMNS (
            score INT PATH '$.score',
            number INt PATH '$.number',
            par INT PATH '$.par'
        )
        ) holeScores
        WHERE holeScores.score = 1
        GROUP by DOC ->> '$.lastName', doc ->> '$.firstName'
        ORDER by DOC ->> '$.date' DESC`

    const query = await session.sql( sql )  4
    await query.execute( (ace) => {
        aces.push(ace)
    })
    session.close()
    return aces
}
```

Query Breakdown `JSON_OBJECT()`

```
SELECT JSON_OBJECT(
        'firstName', doc ->> '$.firstName',
        'lastName', doc ->> '$.lastName',
        'date', doc ->> '$.date',
        'courseName', doc->> '$.course.name',
        'holes', JSON_ARRAYAGG(
            JSON_OBJECT(
                'holeNumber',
                holeScores.number,
                'par',
                holeScores.par
            )
        )
    )
    FROM scores
```

1 2 3 4 5 6

Query Breakdown `JSON_TABLE()`

```
JSON_TABLE( 1
      2 doc, '$.holeScores[*]' 3
            COLUMNS (
            score INT PATH '$.score', 4
            number INt PATH '$.number', 5
            par INT PATH '$.par' 6
        )
        ) holeScores 7
        WHERE holeScores.score = 1 8
      9 GROUP by doc ->> '$.lastName', doc ->> '$.firstName'
      10 ORDER by doc ->> '$.date' DESC`
```

Using Raw SQL Example 2

```
const getAggregateCourseScore = async () => {
    let courses = []
    const session = await pool.getSession()

    const sql = `
      1 WITH aggScores AS
            (SELECT doc ->> '$.course.name' course,
                MIN(score) minScore,
                MAX(score) maxScore,
                number
            FROM scores,
          2 JSON_TABLE( doc, '$.holeScores[*]'
                COLUMNS (
                    score INT PATH '$.score',
                    number INT PATH '$.number'
                )) AS scores
            GROUP BY course, number
            ORDER BY course, number)
      3 SELECT JSON_OBJECT( 'courseName', course, 'bestScore',sum( minScore ) )
      4 FROM aggScores
        GROUP BY course
        ORDER BY course;`

    const query = await session.sql( sql )
    await query.execute( (course) => {
        courses.push(course)
    })
    session.close()
    return courses
}
```

CRUD API

add()

```
app.post('/score', async function(req, res, response){
    const success = await addScore( req.body );
    let msg = { success: success }
    res.send( msg )
});
```

```
const addScore = async ( score ) =>{
    let success = true;
    const session = await pool.getSession()
    const db = session.getSchema(databaseName)
    const collection = db.getCollection(collectionName)
    try{
        await collection.add( score ).execute()
    }
    catch ( e ) {
        success = false
    }
    session.close()
    return success
}
```

CRUD API

modify()

```
app.post('/holeScores', async function(req, res, response){
    const success = await addHoleScores( req.body );
    let msg = { success: success }
    res.send( msg )
});
```

```
const addHoleScores = async ( data ) =>{
    let success = true;
    const session = await pool.getSession()
    const db = session.getSchema(databaseName)
    const collection = db.getCollection(collectionName)
    try{
        await collection.modify( "_id = :id" )
        .set("holeSores", data.holeScores).bind("id", data._id).execute()
    }
    catch ( e ) {
        success = false
    }
    session.close()
    return success
}
```

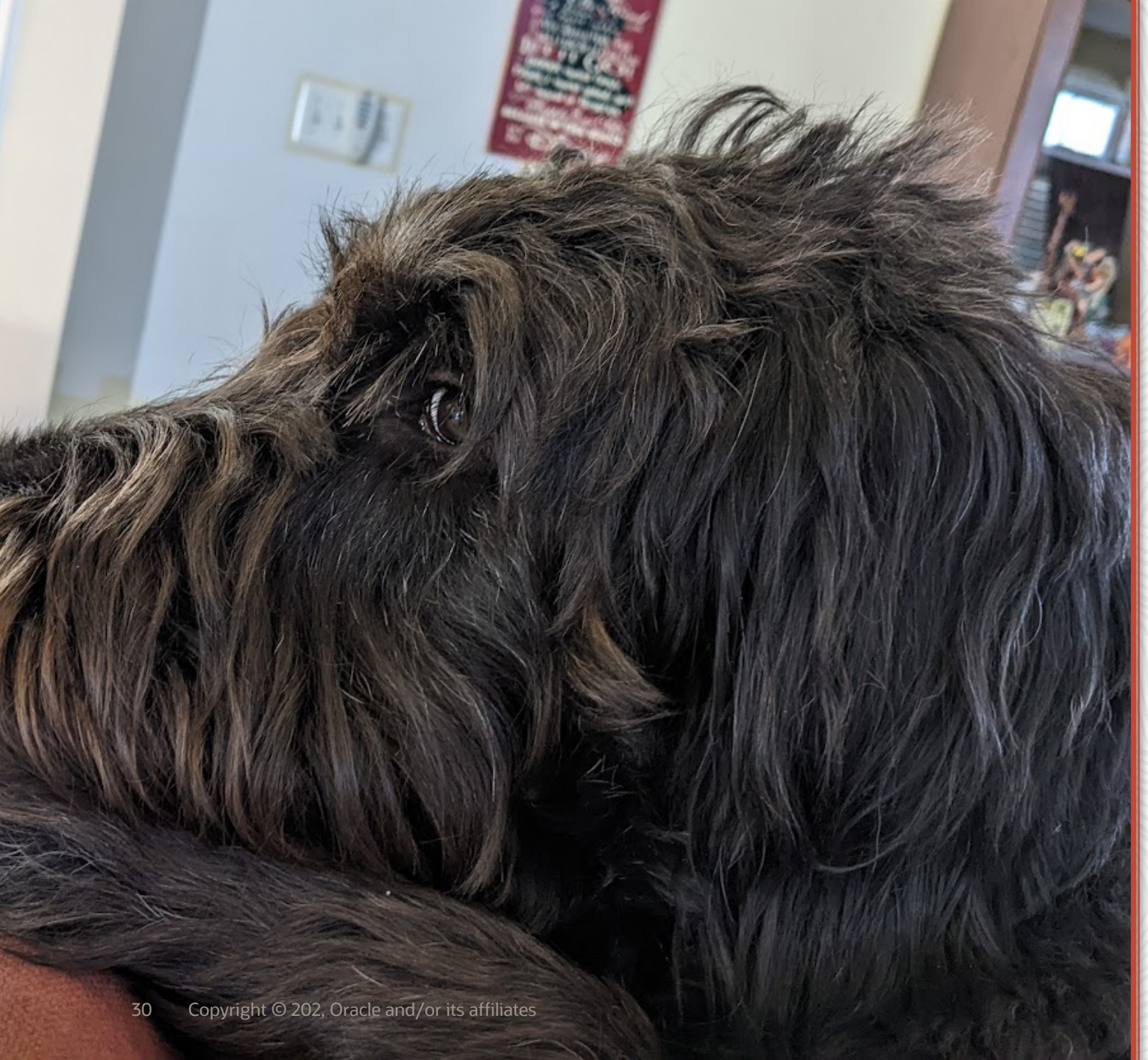
CRUD API

remove()

```
app.get('/removeScore/:id?', async function(req, res, response){
    let msg = {}
  1 if( req.params.id ){
        let success = await removeScore( req.params.id )2
        msg.success = success
    }
    else{
      3 msg.error =  "Please provide a valid id."
    }

    res.send( msg )
});
```

```
const removeScore = async ( id ) =>{
    let success = true;
    const session = await pool.getSession()
    const db = session.getSchema(databaseName)
    const collection = db.getCollection(collectionName)
    try{                    4
        await collection.remove( "_id = :id" ).bind("id", id).execute()
    }                                                      5
    catch ( e ) {
        success = false
    }
    session.close()
    return success
}
```



RECAP

- "NoSQL" vs Relational Databases
- Overview of MySQL Document Store
- Anatomy of MySQL Document Store
- X-Plugin and X-Protocol for connectivity
- Adding Node connector and configuring connection
- Using the CRUD API
- Leveraging the power of raw SQL for ad-hoc queries

Resources

GitHub

- https://github.com/boyzoid/node-document-store-demo

MySQL Document Store Documentation

- https://www.mysql.com/products/enterprise/document_store.html
- https://dev.mysql.com/doc/dev/connector-nodejs/8.0/