

ORACLE



Using MySQL Document Store with Node.js

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 [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

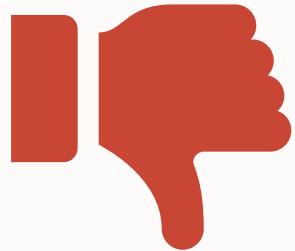


- Faster development time
- Easier to modify the 'schema'
- Simple CRUD API

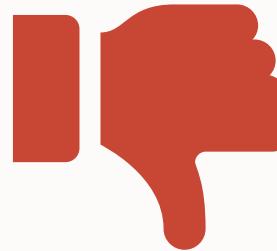
Relational Data



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



- Data is unstructured
- Difficult 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 we
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

```
version: "1.0.0",
description: "",
main: "index.js",
scripts: {
  start: "electron .",
  dev: "rollup -c -w",
  build: "rollup -c"
},
keywords: [],
author: "",
license: "ISC",
devDependencies: {
  "electron": "8.2.1",
  "electron-reload": "1.5.0",
  "concurrently": "5.1.0",
  "@rollup/plugin-commonjs": "11.0.0",
  "@rollup/plugin-node-resolve": "7.0.0",
  "rollup": "1.20.0",
  "rollup-plugin-livereload": "1.3.0",
  "rollup-plugin-svelte": "5.8.3",
  "rollup-plugin-terser": "5.1.2",
  "svelte": "3.21.0"
},
dependencies: {
  "cron": "1.8.2",
  "node-notifier": "7.0.0",
  "rollup-plugin-scss": "2.0.0"
}
```

Anatomy of MySQL Document Store

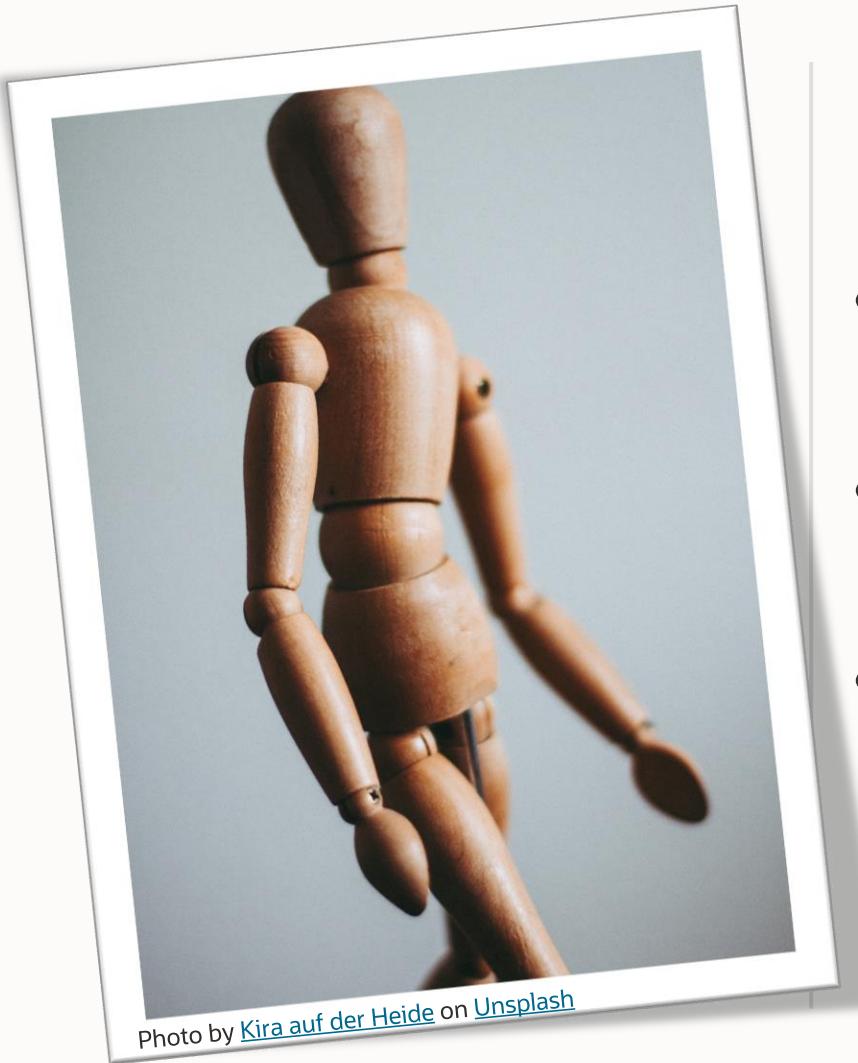


Photo by [Kira auf der Heide](#) on [Unsplash](#)

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

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

** Others if you add indexes.*

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
```

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
```

Connecting to MySQL Document Store with Node

Create Connection Pool

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



Schema Used for this Demo

```
{  
  "_id": "000062f71e9400000000000011129",  
  "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

find()

```
// 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
        .execute((score) => {
            scores.push(score) ⑥
        });
    session.close() ⑦
    return scores
}
```

```
// /list endpoint
app.get('/list/', async (req, res) =>{
    const scores = await listAllScores()
    let msg = { count: scores.length, scores: 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)
        .executes((score) => {
            scores.push(score)
        });
    session.close()
    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) ④
    .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 )
})
```

```
const pool = require('knex')(config)
const getBestScores = async (limit) =>{
  let scores = []
  const session = await pool.getSession()
  const db = session.getSchema(databaseName)
  const collection = db.collection(collectionName)
  await collection.find()
    .fields(['firstName', 'lastName', 'score', 'date', 'course.name as courseName'])
    1 .sort([{score asc}, {date desc}])
    .executed((score) => {
      scores.push(score)
    })
  session.close()
  return scores
}
```

CRUD API

sort() & fields()

```
// 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()
        .fields(['firstName', 'lastName', 'score', 'date', 'course.name as courseName'])5
        .sort(['score asc', 'date desc'])
        .limit( limit )
        .execute((score) => {
            scores.push(score)
        });
    session.close()
    return scores
}
```

```
// /bestScores endpoint
app.get('/bestScores/:limit?', async (req, res) =>{
    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 )
})
```

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 )
})
```

```
const getByScore = async ( score ) =>{
  let scores = []
  const session = await pool.getSession()
  const schema = session.getSchema(databaseName)
  const collection = db.getCollection(collectionName)
  await collection.find(`score = ${score}`)
    .bind(`score`, score)
    .fields(`firstname, lastname as golfer`,
            `score`,
            `date`,
            `course.name as courseName`)
    .sort(`date desc`)
    .execute((err, result) => {
      scores.push(result)
    })
  session.close()
  return scores
}
```

CRUD API

find() & bind()

```
//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
}
```

```
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 )
})
```

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) ③
  await collection.find("lower(lastname) like :lastName")
    .bind('lastName', lastName.toLowerCase()+'%')④
    .sort(['lastName', 'firstName'])
    .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) ③
  await collection.find("lower(lastName) like :lastName")
    .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' )
    .fields(['lastName',
      'firstName',
      'roundAvg(score), 2) as avg'②
      'count(score) as numberOfRounds' ]③
    .sort([ 'lastName', 'firstName'])
    .groupBy(['lastName', 'firstName']) ④
    .execute((score) => {
      scores.push( score )
    })
  session.close()
  return scores
}
```

CRUD API

groupBy()

```
app.get('/getAverageScorePerGolfer/:year?', async (req, res) =>{  
    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' )  
        .fields(['lastName',  
                'firstName',  
                'round(avg(score), 2) as avg',  
                'count(score) as numberofRounds'])  
        .sort([ 'lastName', 'firstName' ])  
        .groupBy(['lastName', 'firstName'])  
        .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 Example 1

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

    ② const sql = `SELECT
        JSON_OBJECT(④
            'firstName', doc ->> '$.firstName',
            'lastName', doc ->> '$.lastName',
            'date', doc ->> '$.date',
            'courseName', doc->> '$.course.name', ⑥
            'holes', JSON_ARRAYAGG( ⑦
                JSON_OBJECT(
                    'holeNumber', holeScores.number,
                    ⑧ 'par', holeScores.par
                )
            )
        )
        FROM scores,
        ⑨ JSON_TABLE(
            ⑩ doc,
            '$.holeScores[*]' ⑪
            COLUMNS (
                ⑫ score INT PATH '$.score',
                number INT PATH '$.number',
                par INT PATH '$.par'
            )
        ) holeScores ⑬
        WHERE holeScores.score = 1 ⑭
        GROUP by doc ->> '$.lastName',
        ⑮ doc ->> '$.firstName'
        ORDER by doc ->> '$.date' DESC` ⑯

    const query = await session.sql( sql ) ⑳
    await query.execute( (ace) => {
        aces.push(ace)
    })
    session.close()
    return aces
}
```



Using Raw SQL Example 1



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

    ② const sql = `SELECT
        COUNT(*) AS hole_count
      FROM
        holes`
```

Using Raw SQL Example 1



```
const query = await session.sql( sql )③
await query.execute( (ace) => {
    aces.push(ace)
})
session.close()
return aces
}
```

Using Raw SQL Example 1

```
2 const sql = `SELECT
  JSON_OBJECT( 4
    'firstName', doc ->> '$.firstName',
    'lastName', doc ->> '$.lastName',
    'date', doc ->> '$.date',
    'courseName', doc->> '$.course.name', 6
    'holes', JSON_ARRAYAGG( 7
      JSON_OBJECT(
        'holeNumber', holeScores.number,
        8 'par', holeScores.par
      )
    )
  )
FROM scores,
```

Using Raw SQL Example 1

```
9 JSON_TABLE(  
10 doc,  
11 '$.holeScores[*]'  
12 COLUMNS (  
13 score INT PATH '$.score',  
14 number INT PATH '$.number',  
15 par INT PATH '$.par'  
16 )  
17 ) holeScores  
18 WHERE holeScores.score = 1  
19 GROUP by doc ->> '$.lastName',  
20 doc ->> '$.firstName'  
21 ORDER by doc ->> '$.date' DESC`  
22
```

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
}
```

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)
```

Using Raw SQL Example 2



```
        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()



```
1 app.post('/score', async function(req, res, response){
  const success = await addScore( req.body );2
  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()❸
  }
  catch ( e ) {
    success = false
  }
  session.close()
  return success
}
```

CRUD API

add()

```
1 app.post('/score', async function(req, res, response){  
2     const success = await addScore( req.body );  
3     let msg = { success: success }  
4     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() ④  
    }  
    catch ( e ) {  
        success = false  
    }  
    session.close()  
    return success  
}
```

CRUD API

modify()



```
1 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("holeScores", data.holeScores).bind("id", data._id).execute()  
    } catch ( e ) {  
        success = false  
    }  
    session.close()  
    return success  
}
```

CRUD API

modify()

```
i 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" )  
    5.set("holeScores", 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 = false
    const session = await pool.getSession()
    const db = session.getSchema(databaseName)
    const collection = db.getCollection(collectionName)
    try{
        ① await collection.remove( "_id = :id" ),bind("id", (id).execute())
    } catch ( e ) {
        success = false
    }
    session.close()
    return success
}
```

CRUD API

remove()

```
app.get('/removeScore/:id?', async function(req, res, response){  
    let msg = {}  
    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()  
    }  
    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/>

MySQL Shell

<https://dev.mysql.com/doc/mysql-shell/8.0/en/>

Me

scott.stroz@oracle.com

