



# Using MySQL Document Store with Node.js

Scott Stroz

MySQL Developer Advocate

# Obligatory "I Love Me" Slide

- MySQL Developer Advocate
- Developer for 20+ years
  - Only constant in that stack has been MySQL
- 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

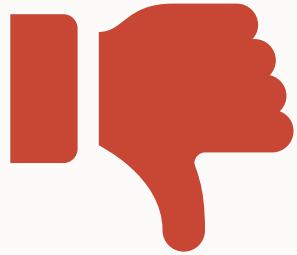


- Faster development time
- Easier to modify the 'schema'
- Simple CRUD API

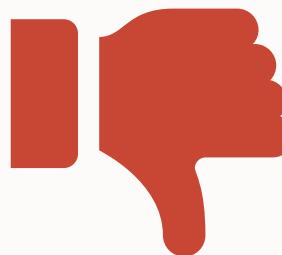


## Relational Data

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



- Data is unstructured
- Difficult to run queries for reporting



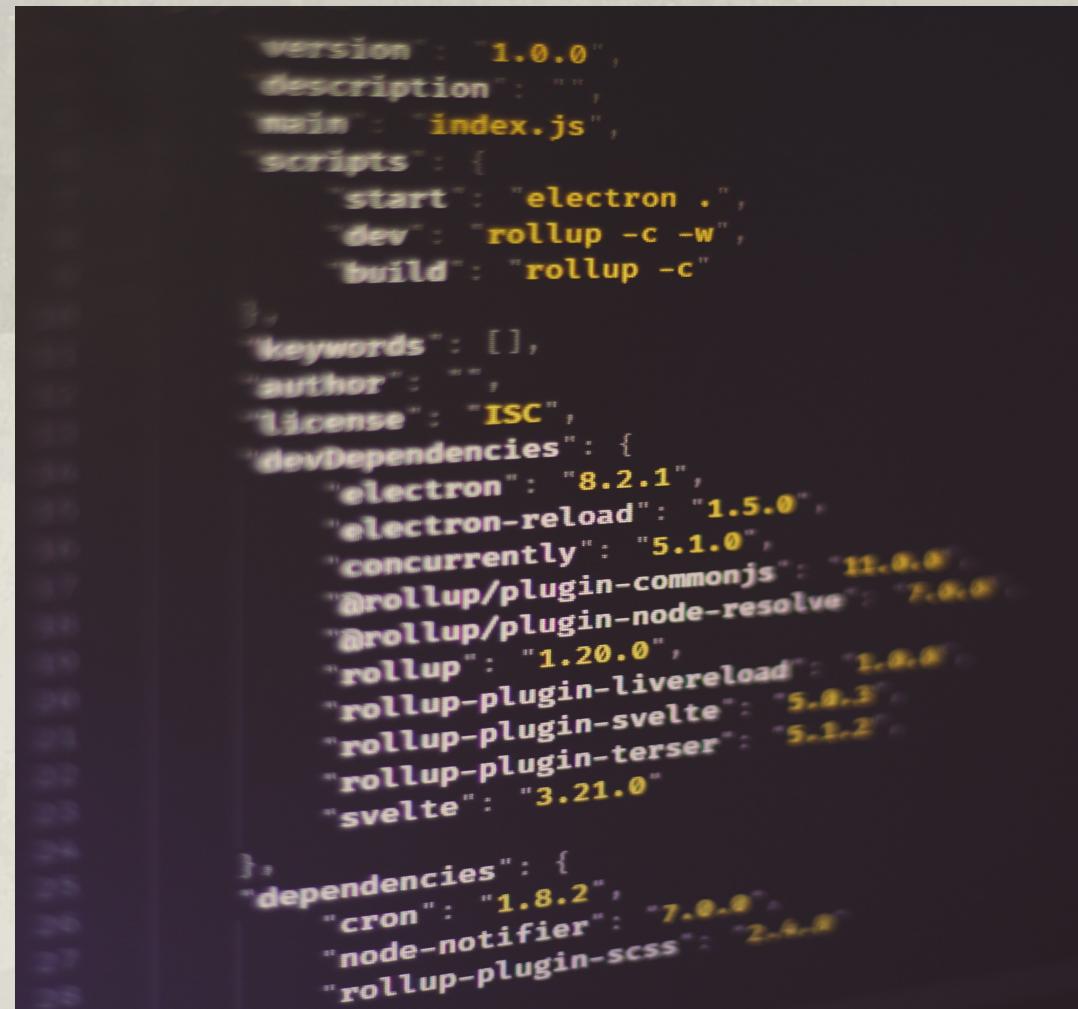
- Slower development time
- As more related data is added, CRUD operations can get more complex



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

# What is MySQL Document Store?

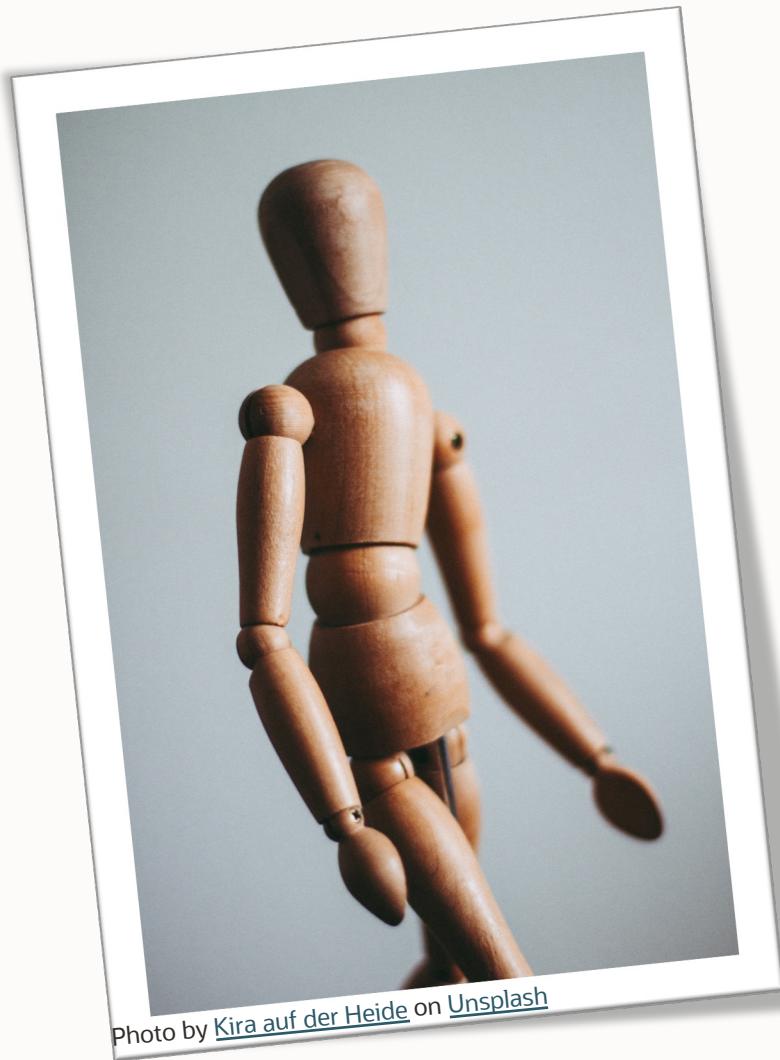
- JSON document storage solution built on top of MySQL
- Stored in MySQL table but abstracted from the user
  - InnoDB storage engine
  - ACID compliant
  - Uses JSON datatype
- Easy to use CRUD API



```
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": "2.4.0"
}
```

Photo by [Ferenc Almasi](#) on [Unsplash](#)

# Anatomy of MySQL Document Store



- **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 (SDKs)
  - 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

# About the Application

- Demo of using a REST API
- Built using Express
- It contains just 2 files
  - **/src/index.js**
    - The main page that handles routing
  - **/src/DocumentStore.js**
    - A JavaScript class that is responsible for communicating with MySQL Document Store.
- Uses Environment variables for database connection information
- The schema looks like this →

```
{  
  "_id": "000063cad9ee000000000000000001",  
  "date": "2022-08-11",  
  "score": 46,  
  "course": {  
    "par": 36,  
    "city": "Charles Town",  
    "name": "Locust Hill Golf Course - Back Nine",  
    "state": "WV"  
  },  
  "lastName": "Smithers",  
  "firstName": "Russell",  
  "holeScores": [  
    {  
      "par": 5,  
      "score": 6,  
      "number": 13,  
      "relationToPar": 1  
    },  
    {  
      "par": 3,  
      "score": 4,  
      "number": 16,  
      "relationToPar": 1  
    }  
  ]  
  ...  
}
```



This Photo by Unknown Author is licensed under [CC BY-NC](#)



# Installing the Connector

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



# Importing the Document Store Class

```
1 import DocumentStore from "./DocumentStore.js";
const docStore = new DocumentStore(2
  process.env.DB_USER,
  process.env.DB_PASSWORD,
  process.env.DB_HOST,
  process.env.DB_PORT,
  process.env.DB_NAME,
  process.env.COLLECTION_NAME
);
```



# Connecting to MySQL Document Store with Node

Import Connector Object in DocumentStore.js

```
import * as mysqlx from '@mysql/xdevapi'
```

# Breaking Down The Constructor

```
constructor(  
    dbUser,  
    dbPassword, ①  
    dbHost,  
    dbPort,  
    dbName,  
    collectionName  
) {  
  
    ② this.#databaseName = dbName  
    this.#collectionName = collectionName  
  
    ③ this.#connectionUrl =  
        `mysqlx:// ${dbUser}:${dbPassword}@${dbHost}:${dbPort}/ ${dbName}`  
  
    ④ this.#pool = mysqlx.getClient(  
        this.#connectionUrl,  
        {  
            pooling: {  
                enabled: true,  
                maxSize: 10,  
                maxIdleTime: 20000,  
                queueTimeout: 5000  
            }  
        }  
    )  
}
```





## Breaking Down The Constructor

```
constructor(  
    dbUser,  
    dbPassword, 1  
    dbHost,  
    dbPort,  
    dbName,  
    collectionName  
) {
```

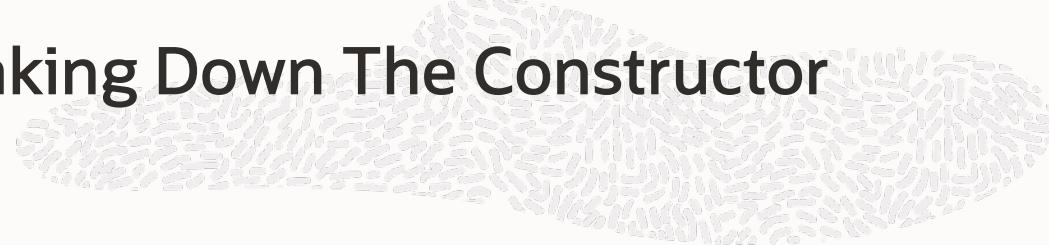


# Breaking Down The Constructor

② `this.#databaseName = dbName`  
`this.#collectionName = collectionName`



# Breaking Down The Constructor



```
3 this.#connectionUrl =  
`mysqlx://${dbUser}:${dbPassword}@${dbHost}:${dbPort}/${dbName}`
```



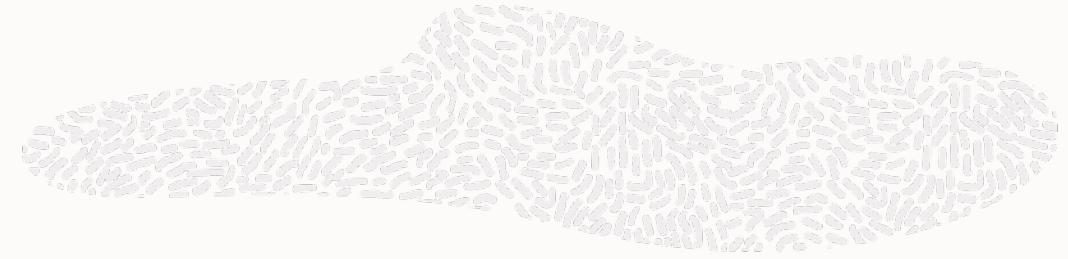


# Breaking Down The Constructor

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

# CRUD API

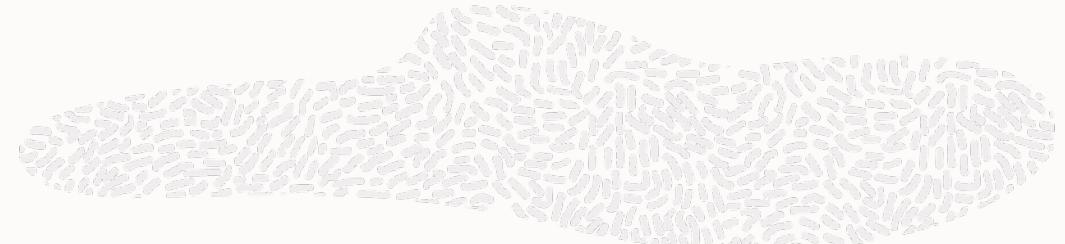
`find()`



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

# CRUD API

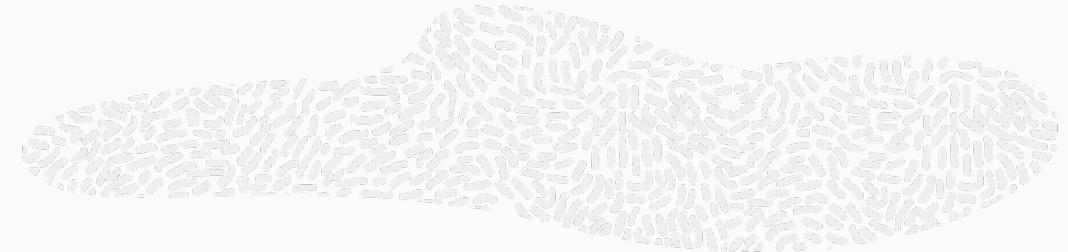
## find()



```
async listAllScores() {  
  2 const session = await this.#pool.getSession()  
  3 const db = session.getSchema(this.#databaseName)  
  4 const collection = db.getCollection(this.#collectionName)  
  5 let results = await collection.find()  
    .execute();  
  6 let data = results.fetchAll()  
  7 session.close()  
  8 return data  
}
```

# CRUD API

## limit() & offset()



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

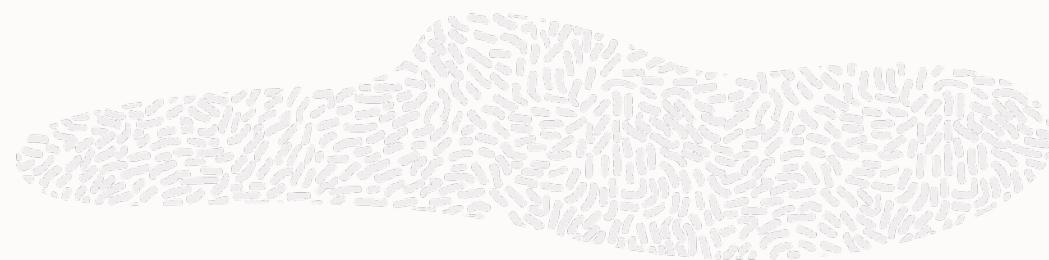
# CRUD API

## limit() & offset()

```
async limitAllScores (limit, offset) {  
    ② if(!offset) offset = 0  
    const session = await this.#pool.getSession()  
    const db = session.getSchema(this.#databaseName)  
    const collection = db.getCollection(this.#collectionName)  
    let results = await collection.find()  
        .limit(limit) ③  
        .offset(offset) ④  
        .execute()  
    let data = results.fetchAll()  
    session.close()  
    return data  
}
```

# CRUD API

sort() & fields()



```
app.get('/bestScores/:limit?', async (req, res) => {
  1 let limit = req.params.limit ? req.params.limit : 50
  const scores = await docStore.getBestScores(limit) 2
  let msg = {count: scores.length, scores: scores}
  res.send(msg)
})
```

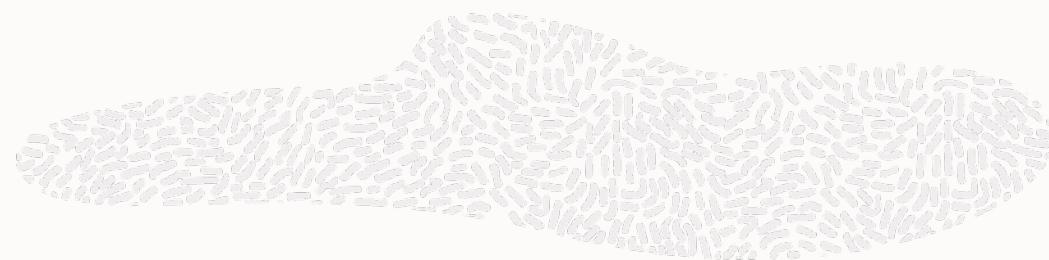
# CRUD API

## sort() & fields()

```
async getBestScores(limit){  
    const session = await this.#pool.getSession()  
    const db = session.getSchema(this.#databaseName)  
    const collection = db.getCollection(this.#collectionName)  
    let results = await collection.find()  
    ③ .fields([  
        'firstName',  
        'lastName',  
        'score',  
        'date',  
        'course'])  
    )  
    ④ .sort(['score asc', 'date desc'])  
    .limit(limit)  
    .execute()  
    let data = results.fetchAll()  
    session.close()  
    return data  
}
```

# CRUD API

## find() with a condition

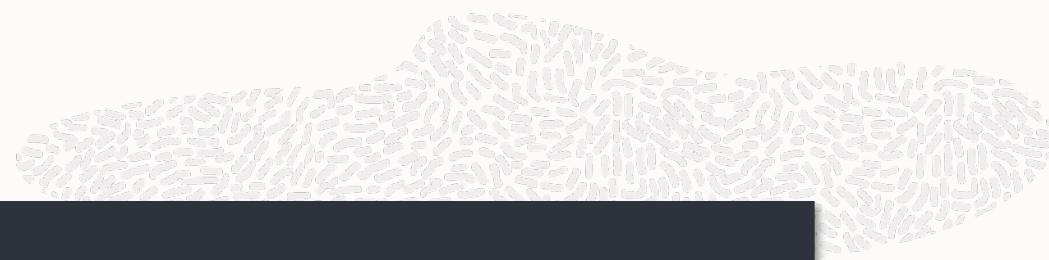


```
app.get('/getRoundsUnderPar', async (req, res) => {
  const scores = await docStore.getRoundsUnderPar() 1
  let msg = {count: scores.length, scores: scores}
  res.send(msg)
})
```

# CRUD API

## find() with a condition

```
async getRoundsUnderPar(){
    const session = await this.#pool.getSession()
    const db = session.getSchema(this.#databaseName)
    const collection = db.getCollection(this.#collectionName)
    let results = await collection.find("score < course.par")
        .fields([
            'firstName',
            'lastName',
            'score',
            'date',
            'course.name as courseName'3
        ])
        .execute()
    let data = results.fetchAll()
    session.close()
    return data
}
```

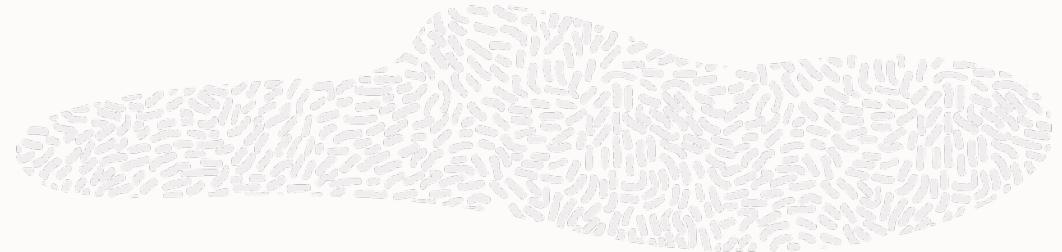


2

3

# CRUD API

## bind()



```
app.get('/getByScore/:score?', async (req, res) => {
  let score = req.params.score ? req.params.score : 361
  const scores = await docStore.getByScore(parseInt(score))
  let msg = {count: scores.length, scores: scores}
  res.send(msg)
})
```

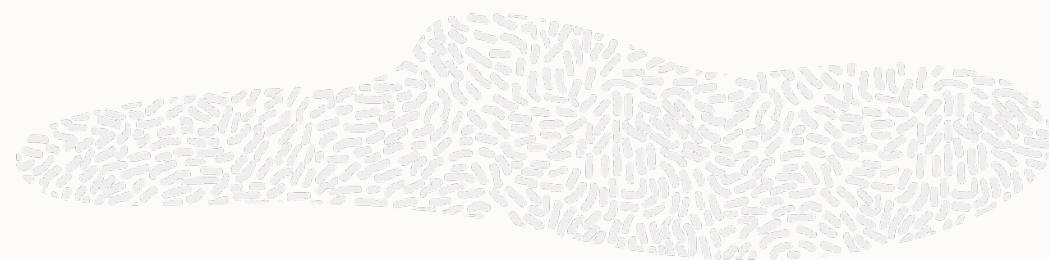
# CRUD API

## bind()

```
async getByScore(score){  
    const session = await this.#pool.getSession()  
    const db = session.getSchema(this.#databaseName)  
    const collection = db.getCollection(this.#collectionName)  
    let results = await collection.find("score = :scoreParam")  
        .bind('scoreParam', score)②  
        .fields([  
            ④ 'concat(firstName, " ", lastName) as golfer',  
            'score', 'date',  
            'course.name as courseName'  
        ])  
        .sort(['date desc'])  
        .execute()  
    let scores = results.fetchAll()  
    session.close()  
    return scores  
}
```

# CRUD API

## bind() with LIKE



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

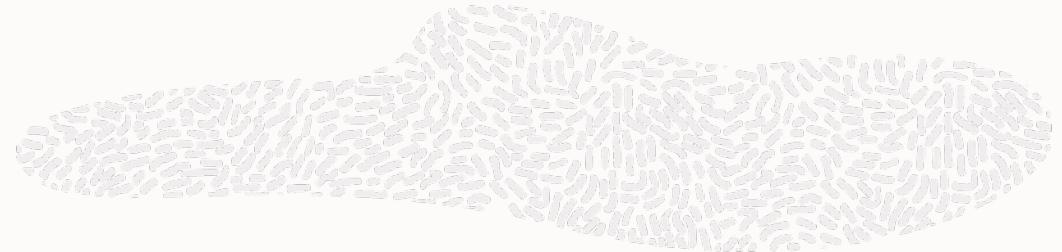
# CRUD API

## bind() with LIKE

```
async getByGolfer(lastName){  
    const session = await this.#pool.getSession()  
    const db = session.getSchema(this.#databaseName)  
    const collection = db.getCollection(this.#collectionName)  
    let results = await collection  
        ②.find("lower(lastName) like :lastNameParam")  
        .bind('lastNameParam', lastName.toLowerCase() + '%')  
        .sort(['lastName', 'firstName'])  
        ③.execute();  
    let data = results.fetchAll()  
    session.close()  
    return data  
}
```

# CRUD API

## groupBy()



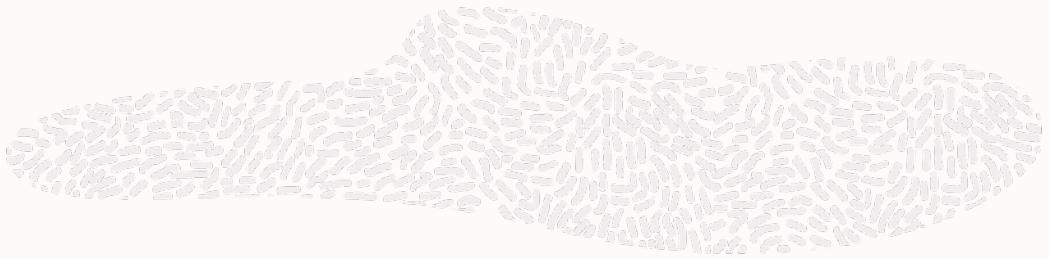
```
app.get('/getCourseScoringData', async (req, res) => {
  const scores = await docStore.getCourseScoringData()
  let msg = {count: scores.length, scores: scores}
  res.send(msg)
})
```

# CRUD API

## groupBy()

```
async getCourseScoringData( ){
    const session = await this.#pool.getSession( )
    const db = session.getSchema(this.#databaseName)
    const collection = db.getCollection(this.#collectionName)
    let results = await collection.find(
        .fields([
            'course.name as courseName',
            ② 'round(avg(score), 2) as avg',
            ③ 'min(cast(score as unsigned)) as lowestScore',
            ④ 'max(cast(score as unsigned)) as highestScore',
            'count(score) as number0fRounds'
        ])
        .groupBy(['course.name']) ①
        .sort('course.name desc')
        .execute()
    let data = results.fetchAll()
    session.close()
    return data
}
```

# Using Raw SQL Example



```
app.get('/getAggregateCourseScore', async (req, res) => {
  const courses = await docStore.getAggregateCourseScore()
  let msg = {courses: courses}
  res.send(msg)
})
```

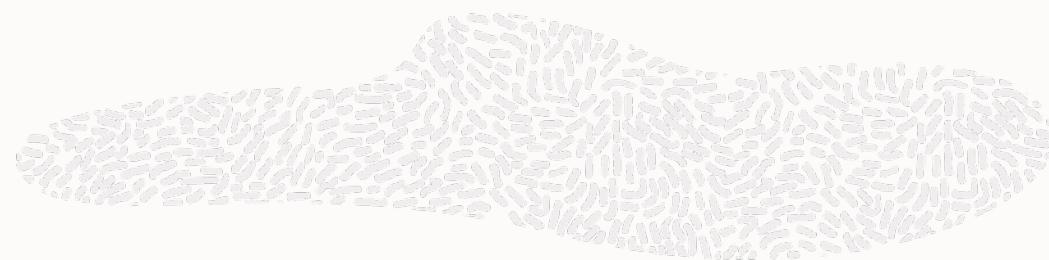
# Using Raw SQL Example

```
async getAggregateCourseScore() {
    const session = await this.#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)
    let results = await query.execute()
    let data = results.fetchAll()
    session.close()
    return data
}
```



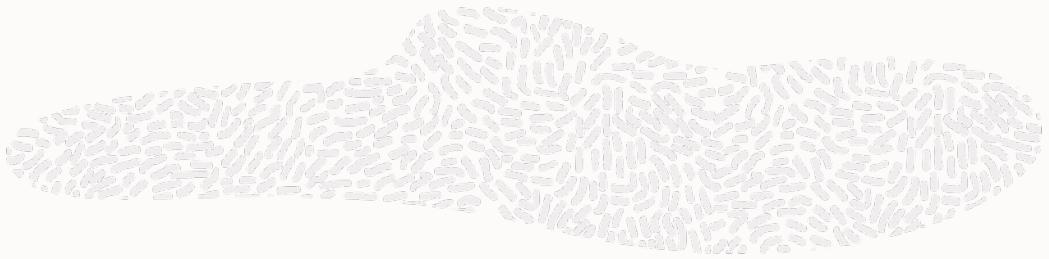
# Using Raw SQL Example



```
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)
```

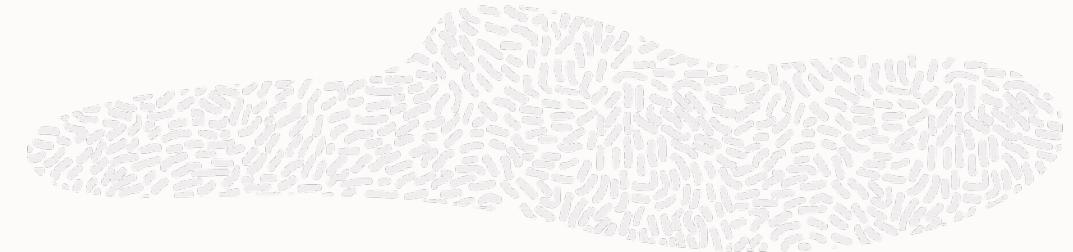
# Using Raw SQL Example



```
3  SELECT JSON_OBJECT('courseName', course, 'bestScore', sum(minScore))  
4  FROM aggScores  
GROUP BY course  
ORDER BY course;  
  
const query = await session.sql(sql)  
5 let results = await query.execute()  
let data = results.fetchAll()  
session.close()  
return data  
}
```

# CRUD API

## add()



```
1 app.post('/score', async function (req, res, response) {  
  const success = await docStore.addScore(req.body);  
  let msg = {success: success} 2  
  res.send(msg)  
});
```

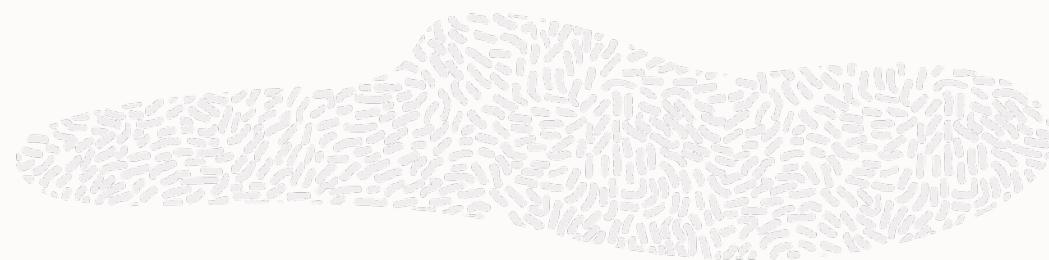
# CRUD API

## add()

```
async addScore(score){  
    let success = true;  
    const session = await this.#pool.getSession()  
    const db = session.getSchema(this.#databaseName)  
    const collection = db.getCollection(this.#collectionName)  
    try {  
        await collection.add(score).execute()③  
    }  
    catch (e) {  
        success = false  
    }  
    session.close()  
    return success  
}
```

# CRUD API

## modify()



```
1 app.post('/holeScores', async function (req, res, response) {  
  const success = await docStore.addHoleScores(req.body);  
  let msg = {success: success} 2  
  res.send(msg)  
});
```

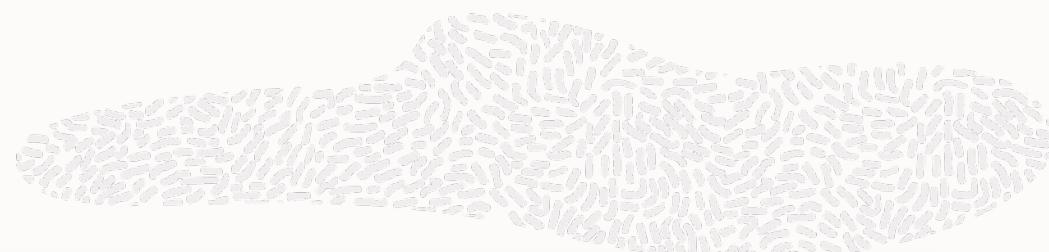
# CRUD API

## modify()

```
async addHoleScores(data){  
    let success = true;  
    const session = await this.#pool.getSession()  
    const db = session.getSchema(this.#databaseName)  
    const collection = db.getCollection(this.#collectionName)  
    try {  
        await collection.modify("_id = :idParam") ③  
            .set("holeScores", data.holeScores) ④  
            .bind("idParam", data._id)  
            .execute()  
    }  
    catch (e) {  
        success = false  
    }  
    session.close()  
    return success  
}
```

# CRUD API

## remove()

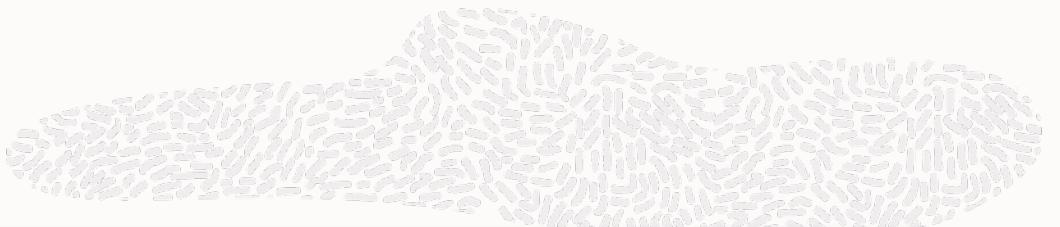


```
app.get('/removeScore/:id?', async function (req, res, response) {
  let msg = {}
  ① if (req.params.id) {
    let success = await docStore.removeScore(req.params.id)
    msg.success = success
  }
  else {
    ② msg.error = "Please provide a valid id."
  }
  res.send(msg)
})
```

# CRUD API

## remove()

```
async removeScore(id){  
    let success = true;  
    const session = await this.#pool.getSession()  
    const db = session.getSchema(this.#databaseName)  
    const collection = db.getCollection(this.#collectionName)  
    try {  
        await collection.remove("_id = :idParam" ) ③  
            .bind("idParam", id)  
            .execute()  
    }  
    catch (e) {  
        success = false  
    }  
    session.close()  
    return success  
}
```



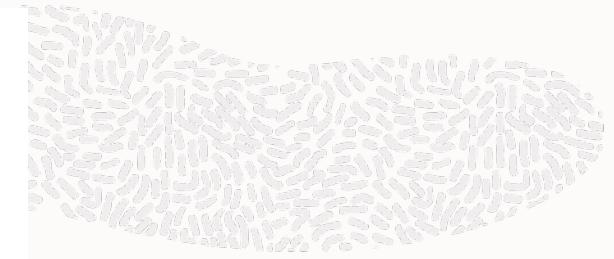
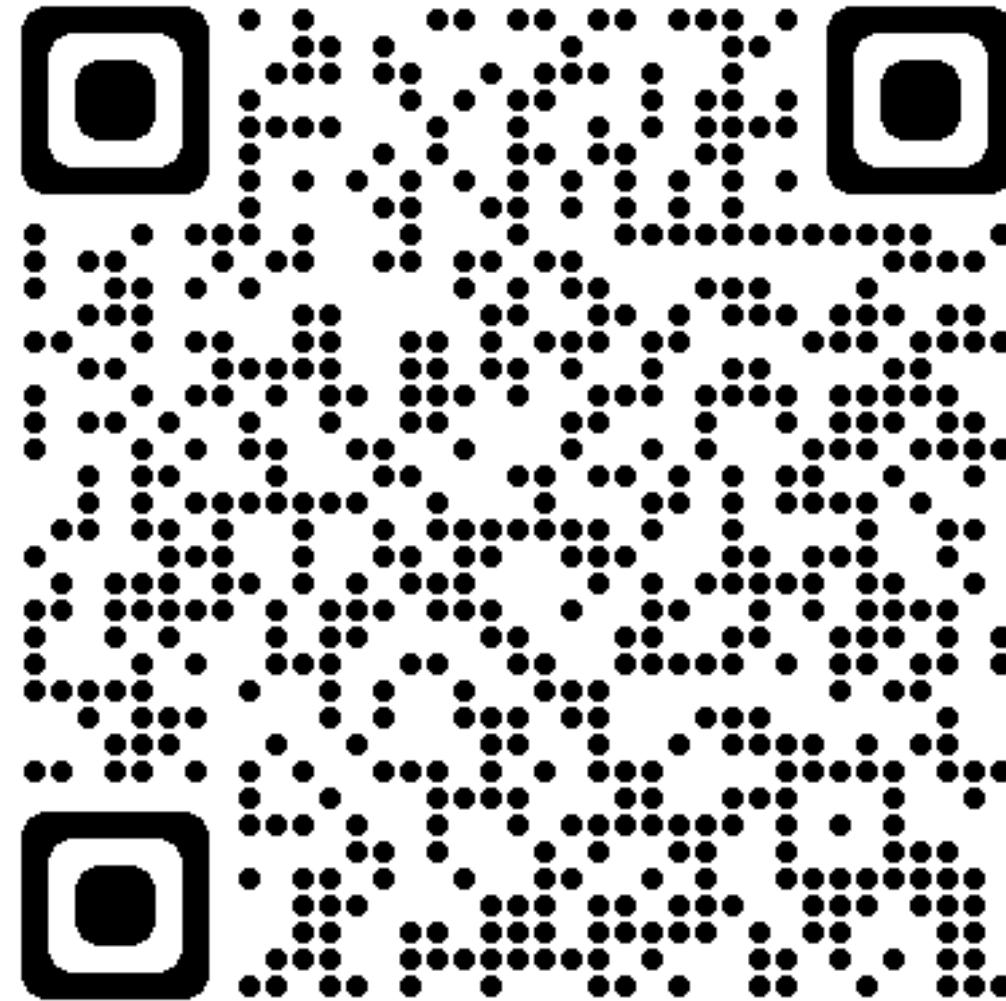


## RECAP

- "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

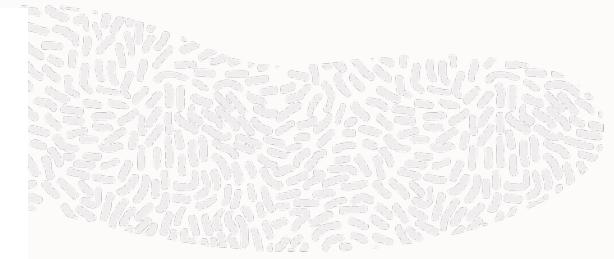
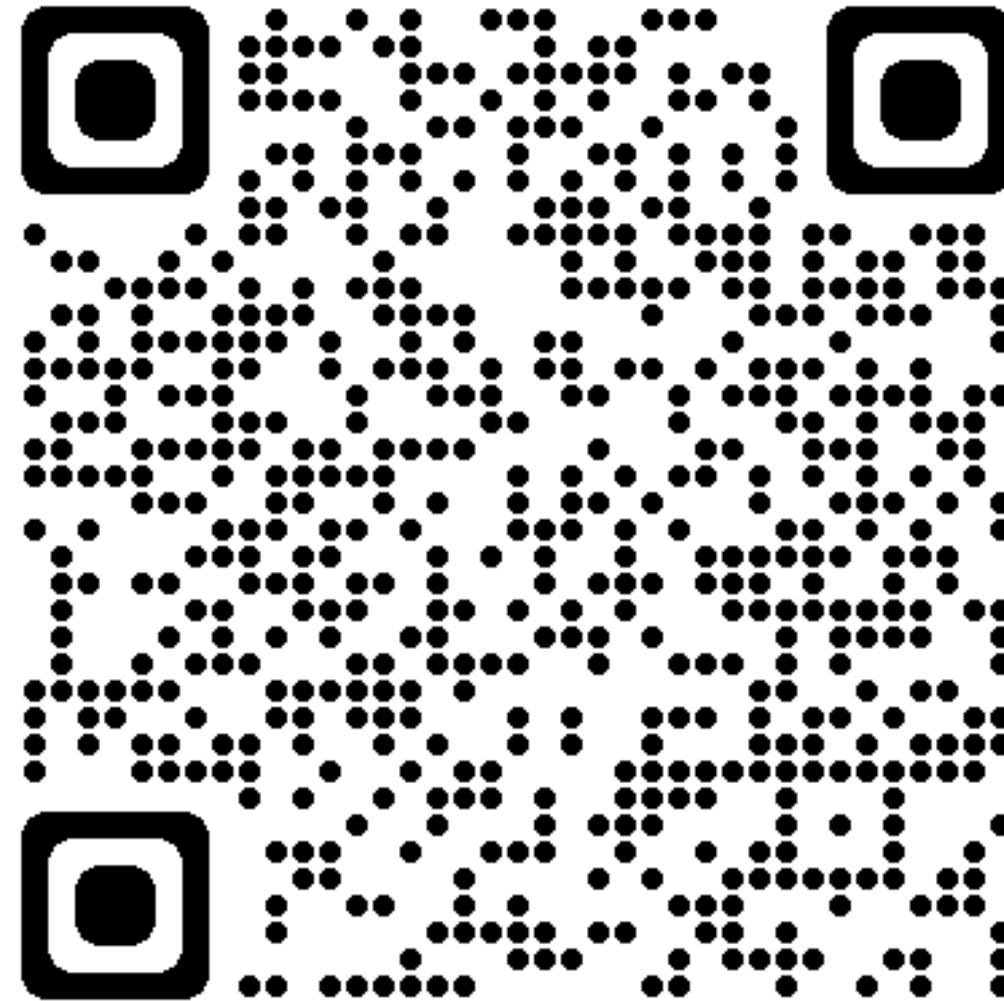
# Resources

GitHub



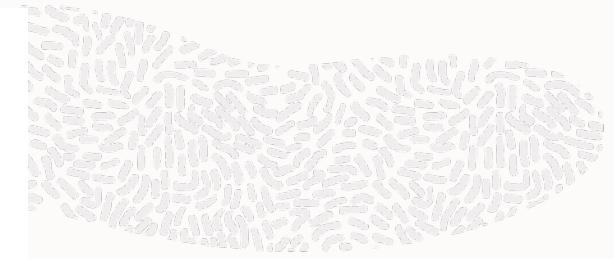
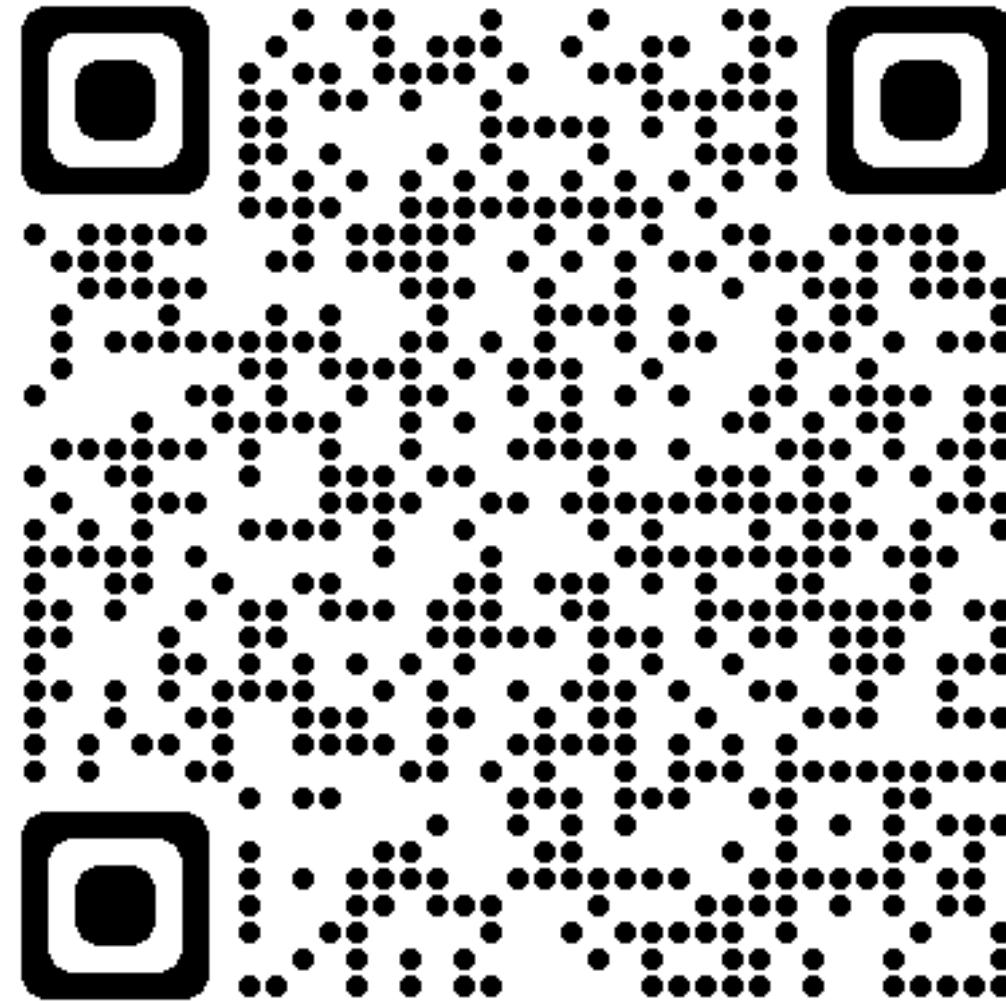
# Resources

Document Store  
Documentation



# Resources

Node Connector  
Documentation



# Contact Me

- **Email**  
[scott.stroz@oracle.com](mailto:scott.stroz@oracle.com)
- **LinkedIn**  
<https://www.linkedin.com/in/scott-stroz/>

- **Twitter**  
<https://twitter.com/boyzoid>



2/15/2023

# Q&A

---

Thank You!

