ORACLE

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

- 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 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

Photo by Ferenc Almasi on Unsplash

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 (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": "000063cad9ee0000000000000001",
  "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
      }
    ]
  ...
}
```

NOT PRODUCTION READY!!

Installing the Connector

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


Photo by Mika Baumeister on Unsplash

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
);
```


Photo by Pat Whelen on Unsplash

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
) {
```


Photo by James Sullivan on Unsplash

Breaking Down The Constructor

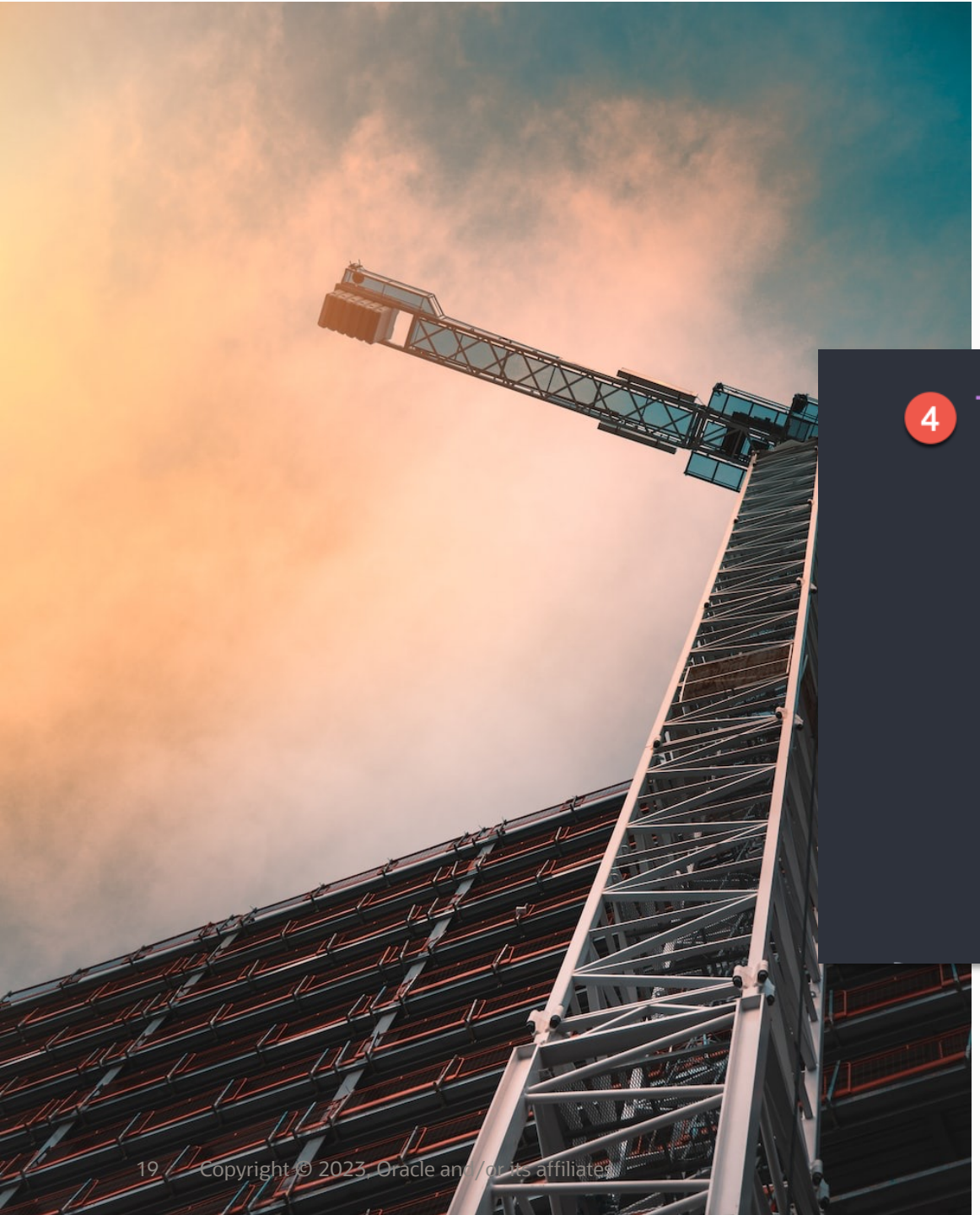
```
2 this.#databaseName = dbName
  this.#collectionName = collectionName
```


Photo by James Sullivan on Unsplash

Breaking Down The Constructor

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

Photo by James Sullivan on Unsplash



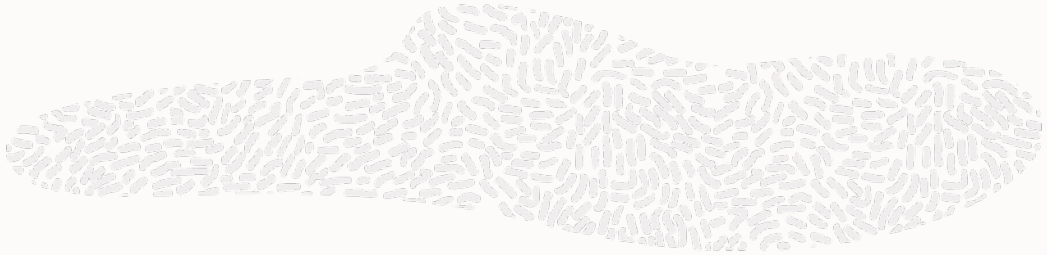
Breaking Down The Constructor

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


Photo by James Sullivan on Unsplash

CRUD API

`find()`



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

1

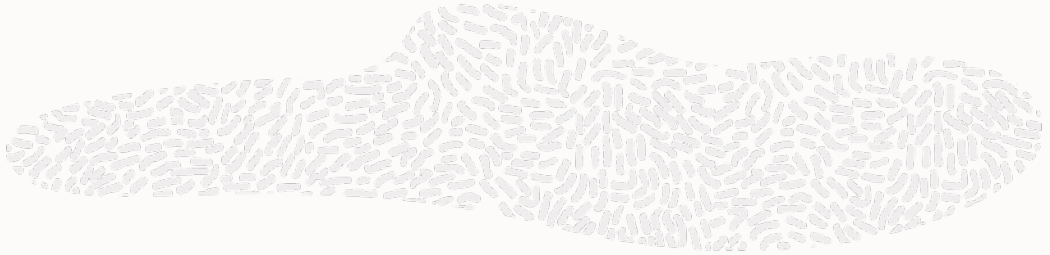
CRUD API

`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(
      req.params.limit,
      req.params.offset
    )
    let msg = {count: scores.length, scores: scores}
    res.send(msg)
})
```

1

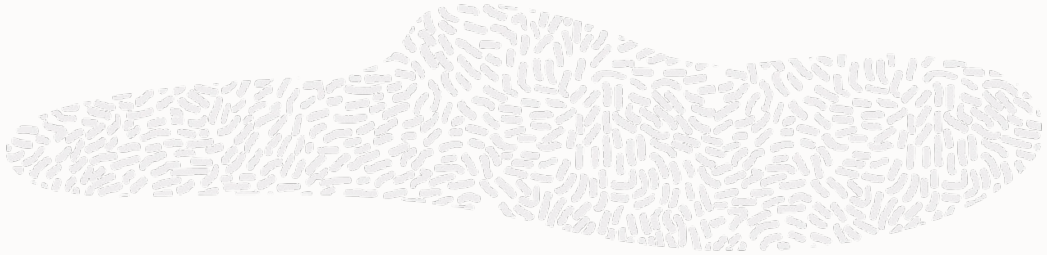
CRUD API

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

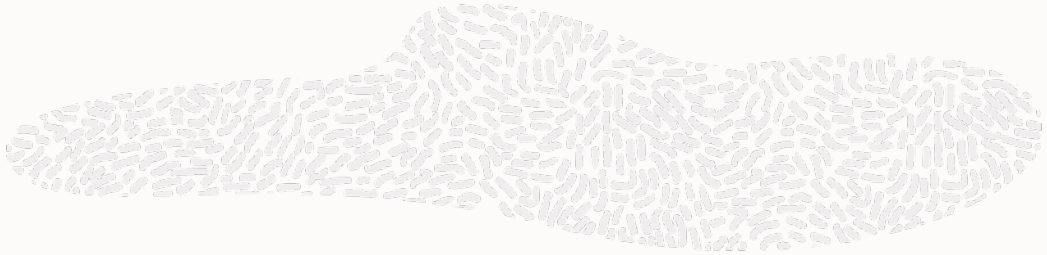
CRUD API

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)
})
```

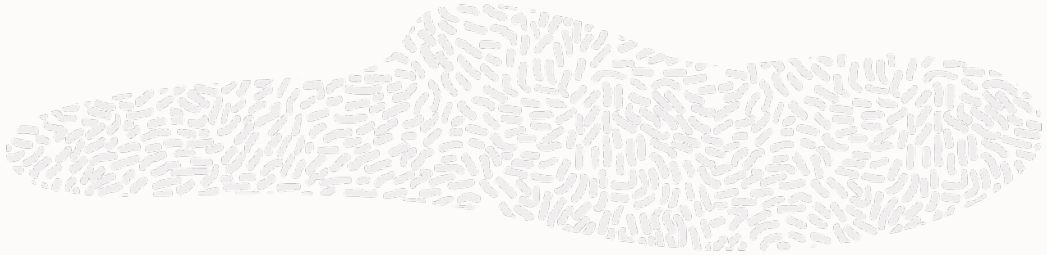
CRUD API

`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'
    ])
    .execute()
  let data = results.fetchAll()
  session.close()
  return data
}
```

CRUD API

bind()



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

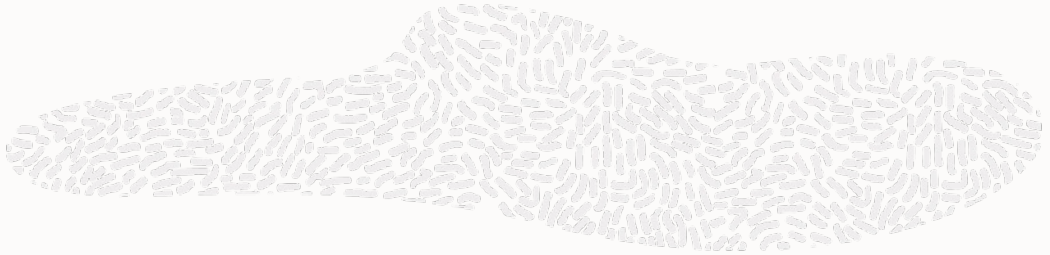
CRUD API

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)
    .fields([
      '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 || ''
  const scores = await docStore.getByGolfer(str)
  let msg = {count: scores.length, scores: scores}
  res.send(msg)
})
```

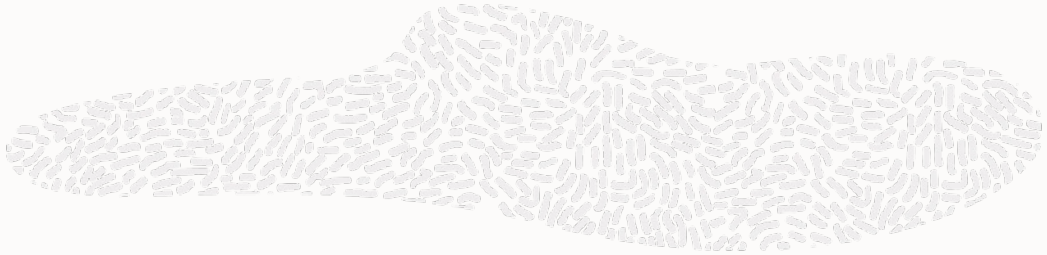
CRUD API

`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)
})
```

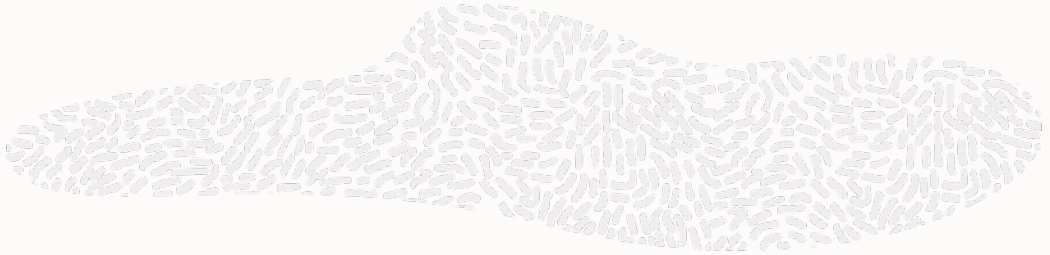
CRUD API

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',
      'round(avg(score), 2)  as avg',
      'min(cast(score as unsigned)) as lowestScore',
      'max(cast(score as unsigned)) as highestScore',
      'count(score) as numberOfRounds'
    ])
    .groupBy(['course.name'])
    .sort('course.name desc')
    .execute()
  let data = results.fetchAll()
  session.close()
  return data
}
```

1. groupBy(['course.name'])
2. 'round(avg(score), 2) as avg',
3. 'min(cast(score as unsigned)) as lowestScore',
4. 'count(score) as numberOfRounds'

Using Raw SQL Example

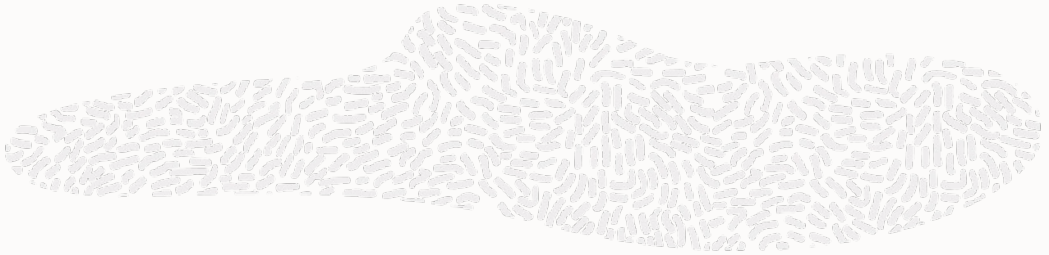


```
app.get('/getAggregateCourseScore', async (req, res) => {
  const courses = await docStore.getAggregateCourseScore()
  let msg = {courses: courses}
  res.send(msg)
})
```

Using Raw SQL Example

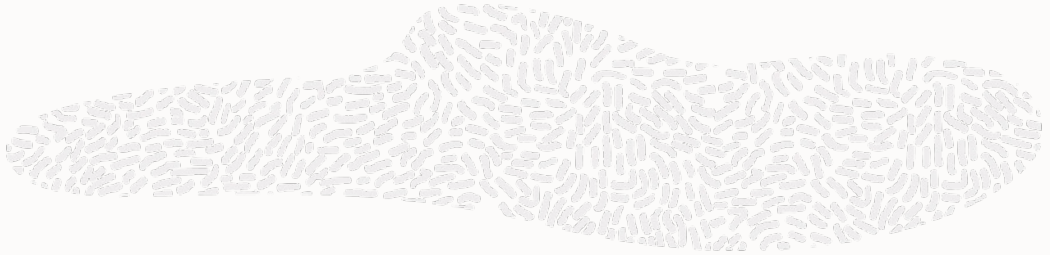
```
async getAggregateCourseScore(){
  const session = await this.#pool.getSession()
  const sql = `
1 WITH aggScores AS
    (SELECT doc ->> '$.course.name' course,
      MIN(score) minScore,
      MAX(score) maxScore,
      number
    FROM scores,
    2 JSON_TABLE(doc, '$.holeScores[*]'
        COLUMNS (
          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)
5 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)
```

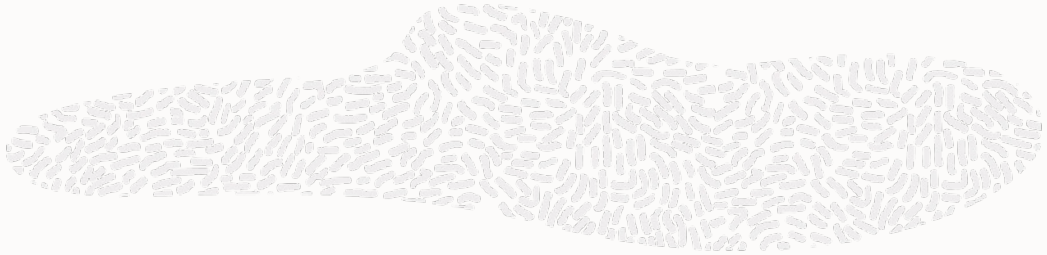
Using Raw SQL Example



```
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()



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

1

2

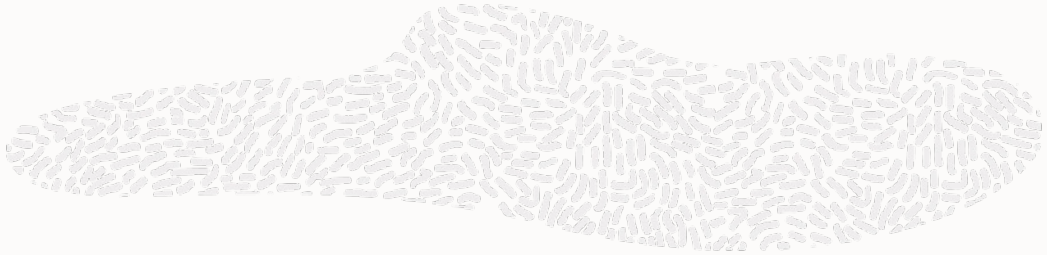
CRUD API

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()
  }
  catch (e) {
    success = false
  }
  session.close()
  return success
}
```

CRUD API

`modify()`



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

1

2

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("holeSores", 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

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/

Me

scott.stroz@oracle.com

Q&A

Thank You!