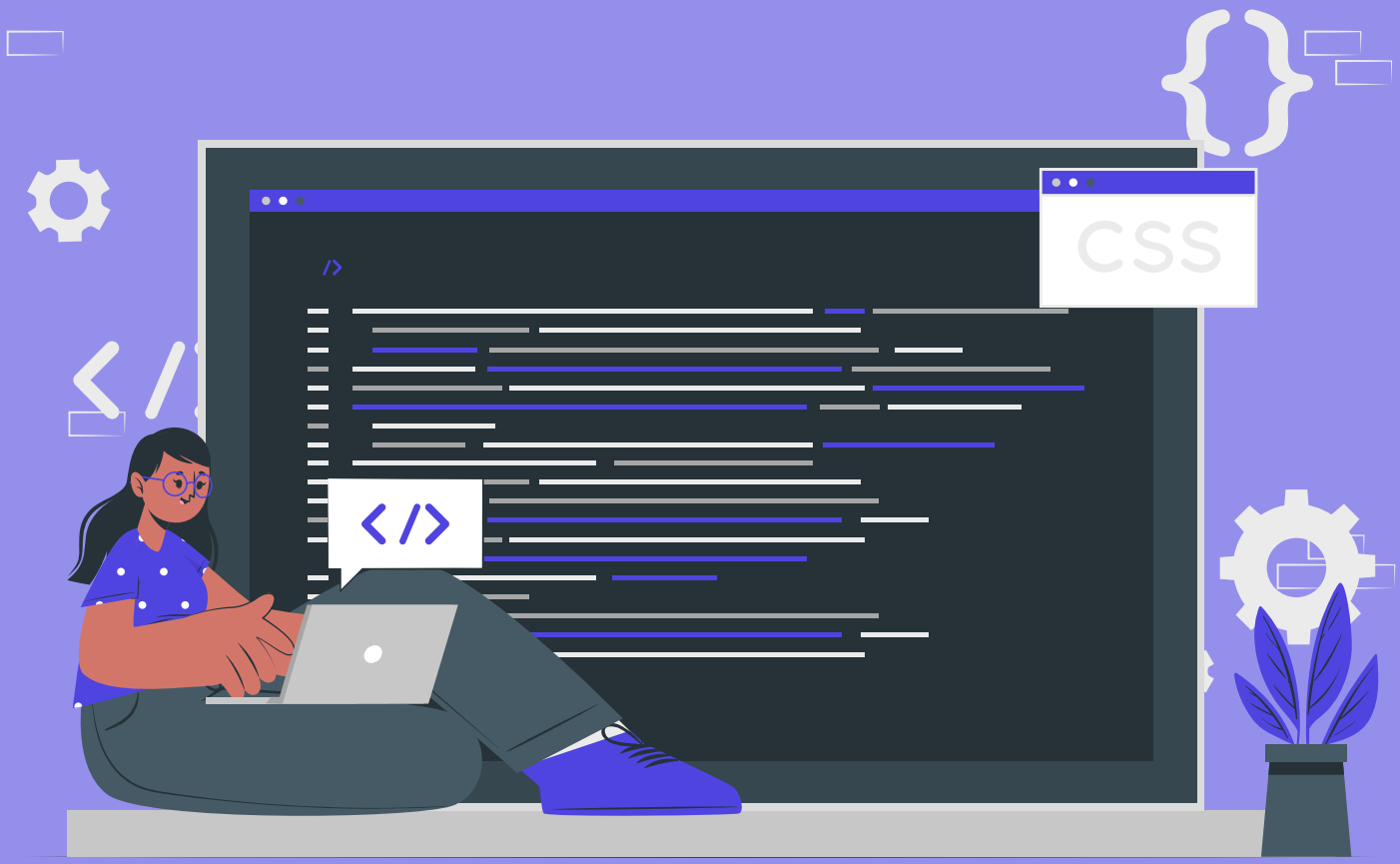
# Create the REST APIs for Products Practice Problems Solutions

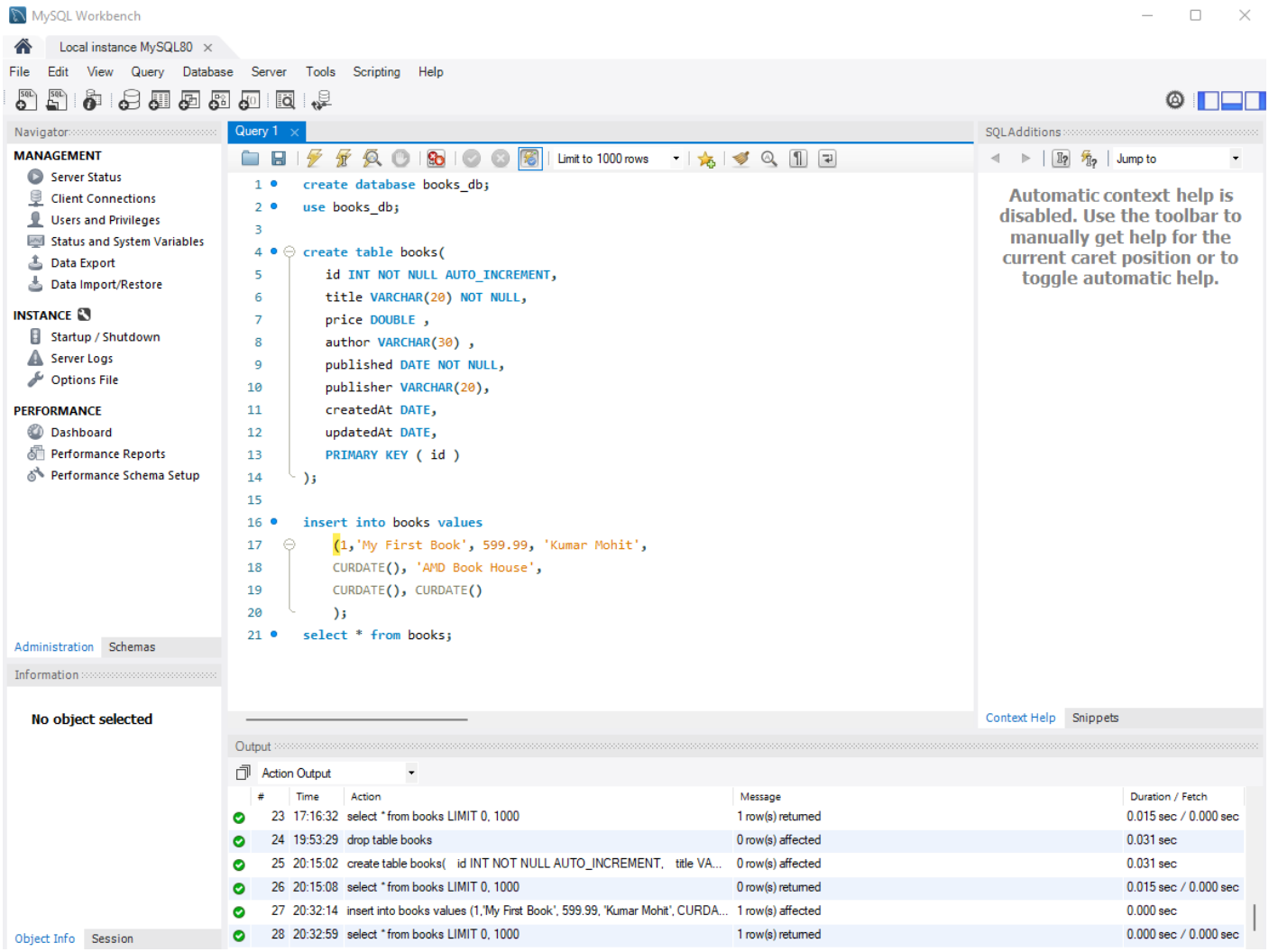



Relevel
by Unacademy

## In our last mini project on books api, let us now use mysql to fetch data from it.

### Step 1: Install MySQL and create database and table

Install MySQL in your machine if not installed from here,
For any issue in installation follow this video .
Once your MySQL workbench is open like this



As shown in picture run the commands in sequence given below:

```
create database books_db;
use books_db;

create table books(
   id INT NOT NULL AUTO_INCREMENT,
   title VARCHAR(20) NOT NULL,
   price DOUBLE ,
   author VARCHAR(30) ,
   published DATE NOT NULL,
   publisher VARCHAR(20),
   createdAt DATE,
   updatedAt DATE,
   PRIMARY KEY ( id )
);
```

insert into books values

```
(1,'My First Book', 599.99, 'Kumar Mohit',
    CURDATE(), 'AMD Book House',
    CURDATE(), CURDATE()
    );
select * from books;
```

Your table will be created and ready to use.

### Step 2: We will start by install mysql2 and sequelize in our application:

Go to command prompt from source directory and run below command

```
npm install mysql2 sequelize
```

### Step 3: Create Books model:

```
const Sequelize = require("sequelize");
// import connection
const db = require("../config/database.js");

// init DataTypes
const { DataTypes } = Sequelize;

// Define schema
const Books = db.define('books', {

    // Define attributes
  title: {
    type: DataTypes.STRING
  },
  price: {
    type: DataTypes.DOUBLE
  },
  author: {
    type: DataTypes.STRING
  },
  published: {
    type: DataTypes.DATEONLY
  },
  publisher: {
    type: DataTypes.STRING
  }
},{
  // Freeze Table Name
  freezeTableName: true
});

// Export model Product
module.exports= Books;
```

### Step 4: Create database config file

```
// import sequelize
const Sequelize = require("sequelize");

// create connection
const db = new Sequelize('books_db', 'root', 'your-mysql-db-password>', {
    host: 'localhost',
    dialect: 'mysql'
});

// export connection
module.exports= db;
```

### Step 5: Update BookController as below:

```
var Book = require("../models/BookModel");


/**
 * Get a list of all the Books
 */
const getAllBooks = async (req, res) => {
   try {
       const books = await Book.findAll();
       res.send(books);
   } catch (error) {
      console.error(error.message);
      res.status(500).json({message: error.message})
   }
}

/**
 * Get a book by it's id
 */
const getBookById = async (req, res) => {
   try {
       const book = await Book.findAll({
           where: {
               id: req.params.id
           }
       });
```

```
        if(book.length > 0){
          res.send(book[0]);
        } else {
          res.status(404).json({message: `Book with id ${req.params.id} not
found`})
        }
    } catch (error) {
       console.error(error.message);
       res.status(500).json({message: error.message})
    }
}

/**
 * Create and save a new Book
 */
const addBook = async (req, res) => {

    // Validations
    if (!req.body.title) {
       res.status(400).send({
             message: "Title of the book can't be empty !"
       })
       return;
    }

    try {
        const book = await Book.create(req.body);
        console.log(book);
        res.status(201).json({
            "message": "Book Created",
            book
        });
    } catch (error) {
       console.error(error.message);
       res.status(500).json({message: error.message})
    }
}

/**
 * Update an existing book based on it's id
 */
const updateBookById = async (req, res) => {

    // Validations
    if (!req.body.title) {
```

```
      res.status(400).send({
            message: "Title of the book can't be empty !"
      })
      return;
   }

   try {
       const book = await Book.update(req.body, {
           where: {
               id: req.params.id
           }
       });
       if(book[0] === 0){
         res.status(404).json({message: `Book with id ${req.params.id} not
found`})
       }
       else {
         res.json({
            "message": `Book Id: ${ req.params.id } Updated`
         });
       }

   } catch (error) {
      console.error(error.message);
      res.status(500).json({message: error.message})
   }
}

/**
 * Delete an existing book based on it's id
 */
const deleteBookById = async (req, res) => {
   try {
       await Book.destroy({
           where: {
               id: req.params.id
           }
       });
       res.json({
           "message": `Book Id: ${ req.params.id } Deleted`
       });
   } catch (error) {
      console.error(error.message);
      res.status(500).json({message: error.message})
   }
```

```
}

module.exports = { getAllBooks, getBookById, addBook, updateBookById,
deleteBookById }
```

### Step 6: Update route accordingly:

```
const express = require('express');
const router = express.Router();
const bookController = require("../controllers/BookController")

//Route for the POST request to add a book
router.post("/", bookController.addBook);

//Route for the GET request to fetch all the books details
router.get("/", bookController.getAllBooks);

//Route for the GET request to fetch a book based on the id
router.get("/:id", bookController.getBookById);

//Route for the PUT request to update a book based on the id
router.put("/:id", bookController.updateBookById);

//Route for the DELETE request to delete a book based on the id
router.delete("/:id", bookController.deleteBookById);


module.exports = router;
```

### Step 7: Update index.js file also as show below:

```
const express = require('express');
const bodyParser = require('body-parser');
const db = require("./config/database");
const bookRoutes = require('./routes/BookRoutes')

// initiaizing express
const app = express();
```

```
* Using the body-parser middleware
*
* Using for parsing the request.
* Parsing the request of the type json and convert that to object
* */
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());

const connectDB = async () => {
   await db.authenticate();
}

// Testing database connection
try {
   connectDB();
   console.log('Connection has been established successfully.');
} catch (error) {
   console.error('Unable to connect to the database:', error);
}



app.get('/', (req, res)=> {
   res.send("This API is working")
});

// book routes to handle all api starting with /books
app.use('/books', bookRoutes);

app.listen(8080, () => {
   console.log('Server started on Port 8080...');
});
```

For reference check the GitHub code link here.