Name of Student: Pushkar Sane			
Roll Number: 45		**Lab Assignment Number: 7**	
Title of Lab Assignment: Create an application to establish a connection with the MySQL database and perform basic database operations on it (student db consisting roll no, name, address), insert 10 records, update a particular student’s record, delete a record.			
DOP: 16-10-2023		**DOS: 17-10-2023**	
CO Mapped: **CO2**	**PO Mapped:** **PO5, PSO1**	**Faculty Signature:**	**Marks:**

Practical No. 7

Aim: Create an application to establish a connection with the MySQL database and perform basic database operations on it (student db consisting roll no, name, address), insert 10 records, update a particular student’s record, delete a record.

Description:

- **MySQL Server:**
  - MySQL is an open-source relational database management system used to store and manage data in a structured manner.

- **Node.js MySQL Driver:**
  - To connect to MySQL from Node.js, you'll need a Node.js MySQL driver. The `mysql` package is a commonly used choice for this purpose. You can install it using `npm`.

    ```bash
    npm install mysql
    ```

- **Code File (e.g. `app.js`):**
  - You'll write your Node.js code in a JavaScript file. This file will contain the logic for connecting to the database, executing queries, and handling the results.

- **Connection Configuration:**
  - Configure your MySQL connection by specifying the following details:
  - Host: The MySQL server's hostname or IP address.
  - User: The MySQL username with appropriate privileges.
  - Password: The password for the MySQL user.
  - Database: The name of the database you want to connect to.

- **Creating a Connection:**
  - Use the MySQL driver to create a connection to the MySQL database. This connection object will be used to perform database operations

    ```javascript
    const mysql = require('mysql');
    const connection = mysql.createConnection({
    ```

```
host: 'localhost',
user: 'your_mysql_username',
password: 'your_mysql_password',
database: 'your_database_name',
});
```

- **Handling Connection Events:**
  - You should handle events related to the database connection, such as errors and successful connections.

```javascript
connection.connect((err) => {
if (err) {
console.error('Error connecting to MySQL: ' + err.stack);
return;
}
console.log('Connected to MySQL as id ' + connection.threadId);
});
```

- **Performing Database Operations:**
  - You can use the `connection` object to execute SQL queries for various database operations, such as inserting, updating, and deleting records, as well as retrieving data (SELECT).

- **Error Handling:**
  - Implement error handling to gracefully deal with any issues that may arise during database operations or the connection process.

- **Closing the Connection:**
  - After you've completed your database operations, make sure to close the connection to free up resources and maintain security.

```javascript
connection.end((err) => {
if (err) {
console.error('Error closing the connection: ' + err.stack);
return;
}
console.log('MySQL connection closed.');
});
```

- We can create a Node.js application that connects to a MySQL database seamlessly. This setup is essential for building web applications, APIs, and other software that require database interaction.

Code:

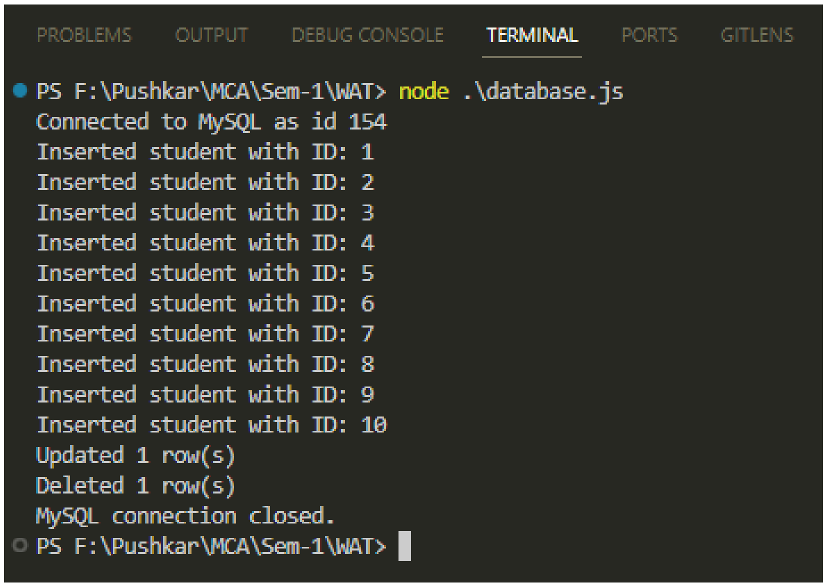
Database.js

```
const mysql = require('mysql2');
// Create a connection to the MySQL database
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'root123',
  database: 'student',
  connectionLimit : 10
});
// Connect to the MySQL server
connection.connect((err) => {
  if (err) {
    console.error('Error connecting to MySQL: ' + err.stack);
    return;
  } else {
    console.log('Connected to MySQL as id ' + connection.threadId);
  }
});
for (let i = 1; i <= 10; i++) {
  const student = {
    rollno: i,
    name: `student ${i}`,
    address: `Address ${i}`,
  };
connection.query('INSERT INTO students SET ?', student, (error, results) => {
  if (error) throw error;
  console.log(`Inserted student with ID: ${results.insertId}`);
  });
}
// Update a particular student's record
const updatedStudent = {
  name: 'Updated Student',
  address: 'Updated Address',
};
connection.query(
  'UPDATE students SET ? WHERE rollno = ?',
  [updatedStudent, 1],
  (error, results) => {
    if (error) throw error;
    console.log(`Updated ${results.affectedRows} row(s)`);
  }
);
// Delete a record (student with rollno 2)
connection.query('DELETE FROM students WHERE rollno = ?', 2, (error, results) => {
if (error) throw error;
  console.log(`Deleted ${results.affectedRows} row(s)`);
});
// Close the connection when done
connection.end((err) => {
  if (err) {
    console.error('Error closing the connection: ' + err.stack);
    return;
  }
console.log('MySQL connection closed.');
});
```

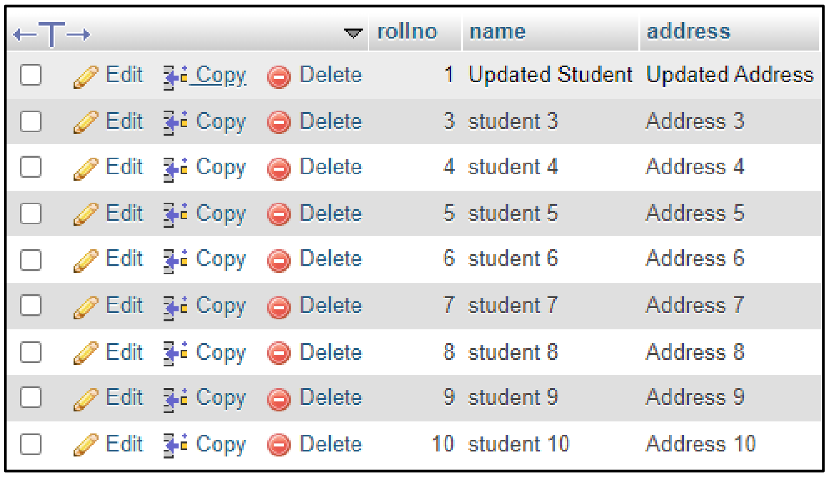
MySQL Queries

```
CREATE DATABASE student;
USE student;
CREATE TABLE students (
    rollno INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    address VARCHAR(255)
);
select * from students;
```

Output:



```
PS F:\Pushkar\MCA\Sem-1\WAT> node .\database.js
Connected to MySQL as id 154
Inserted student with ID: 1
Inserted student with ID: 2
Inserted student with ID: 3
Inserted student with ID: 4
Inserted student with ID: 5
Inserted student with ID: 6
Inserted student with ID: 7
Inserted student with ID: 8
Inserted student with ID: 9
Inserted student with ID: 10
Updated 1 row(s)
Deleted 1 row(s)
MySQL connection closed.
PS F:\Pushkar\MCA\Sem-1\WAT>
```



←T→	rollno	name	address
☐ Edit Copy Delete	1	Updated Student	Updated Address
☐ Edit Copy Delete	3	student 3	Address 3
☐ Edit Copy Delete	4	student 4	Address 4
☐ Edit Copy Delete	5	student 5	Address 5
☐ Edit Copy Delete	6	student 6	Address 6
☐ Edit Copy Delete	7	student 7	Address 7
☐ Edit Copy Delete	8	student 8	Address 8
☐ Edit Copy Delete	9	student 9	Address 9
☐ Edit Copy Delete	10	student 10	Address 10

Conclusion:

Connecting MySQL to Node.js using Visual Studio Code (VSCode) is essential for building database-driven applications. By configuring the connection, handling events, and performing database operations, you can create robust, data-driven software efficiently and securely.