**Final Project: Teacher’s Portal**

**Richmond Ankomah**

Yanilda Peralta

December 21, 2022

**Table of Contents**

[List of Figures 3](#_Toc122598790)

[MySQL To-Do List 4](#_Toc122598791)

[Creating a Database 4](#_Toc122598792)

[Creating and Populating Tables 4](#_Toc122598793)

[Stored Procedure 5](#_Toc122598794)

[Python MySQL 6](#_Toc122598795)

[1. Default Parameter Values 6](#_Toc122598796)

[2. Running portalServer.py 6](#_Toc122598797)

[3. Add Student Link 6](#_Toc122598798)

[4. Add Student Function 7](#_Toc122598799)

[Extra Bonus 9](#_Toc122598800)

[5. Extra Bonus: addCourse () Method in portalDatabase and portalServer 9](#_Toc122598801)

[6. Extra Bonus: searchStudent () in portalDatabase and portalServer 11](#_Toc122598802)

[Appendices 15](#_Toc122598803)

[Appendix 1: Confirmation Upon Adding a New Student 15](#_Toc122598804)

[Appendix 2: Confirmation Upon Adding a New Course 15](#_Toc122598805)

[Appendix 3: List of Students with Grades from The Students Table as Seen from MySQL Workbench 16](#_Toc122598806)

# List of Figures

[Figure 1 4](https://d.docs.live.net/b93100873fe81ad5/Documents/New%20Guy%20Final/teachers_portal/project_report.docx#_Toc122598776)

[Figure 2 5](https://d.docs.live.net/b93100873fe81ad5/Documents/New%20Guy%20Final/teachers_portal/project_report.docx#_Toc122598777)

[Figure 3 6](#_Toc122598778)

[Figure 4 6](#_Toc122598779)

[Figure 5 8](#_Toc122598780)

[Figure 6 8](#_Toc122598781)

[Figure 7 8](#_Toc122598782)

[Figure 8 9](#_Toc122598783)

[Figure 9 10](#_Toc122598784)

[Figure 10 10](#_Toc122598785)

[Figure 11 10](#_Toc122598786)

[Figure 12 13](#_Toc122598787)

[Figure 13 14](#_Toc122598788)

[Figure 14 14](#_Toc122598789)

# MySQL To-Do List

## Creating a Database

The SQL script below is used to create the teacher’s portal database

CREATE DATABASE teachers\_portal;

## Creating and Populating Tables

The script below creates a table with the name “students” with the provided attributes and sets “studentId” as the primary key. The second part of the script populates the table with the values provided.

CREATE TABLE `students` (

`studentId` int(11) NOT NULL,

`studentName` varchar(45) NOT NULL,

`enrolledInCourseID` int(11) NOT NULL DEFAULT 1,

`grade` float DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

INSERT INTO `students` (`studentId`, `studentName`, `enrolledInCourseID`, `grade`) VALUES

(1, 'Maria Jozef', 1, 90),

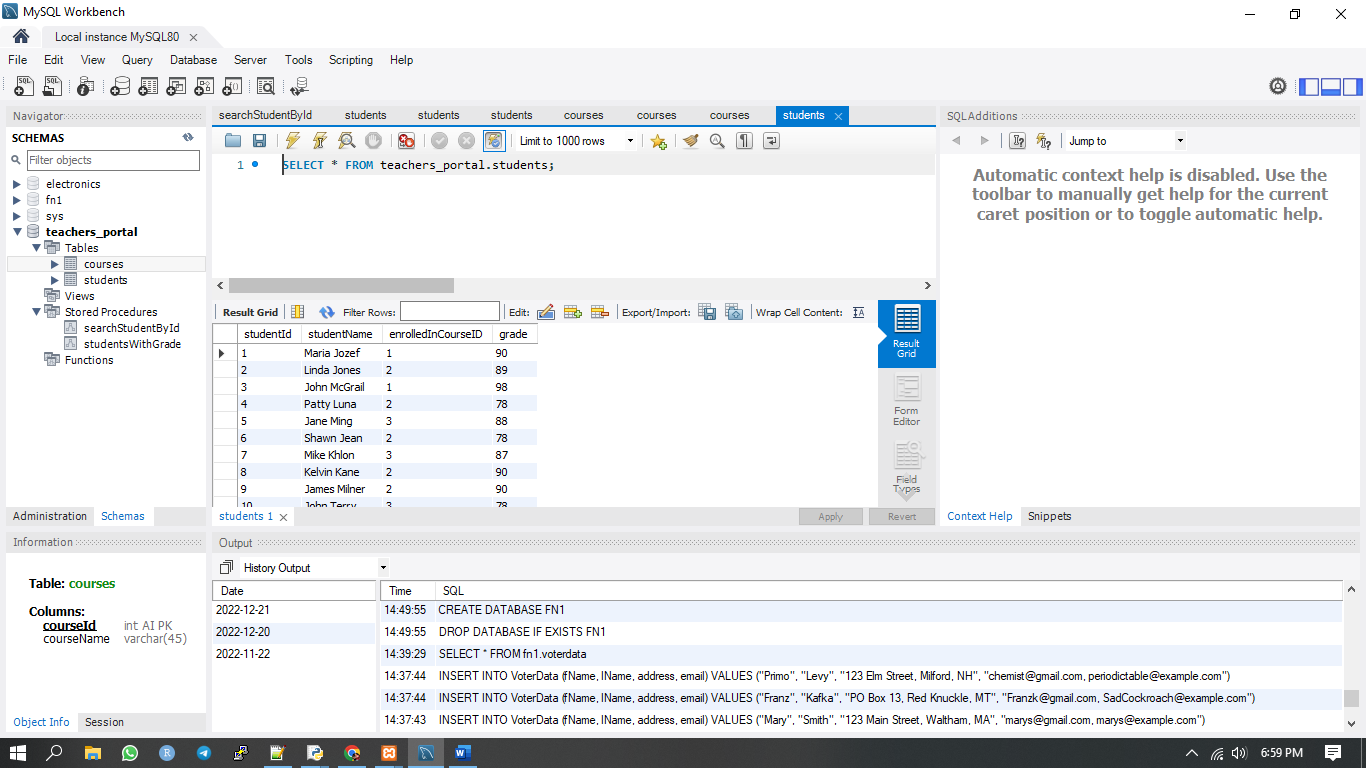
(2, 'Linda Jones', 2, 89),

(3, 'John McGrail', 1, 98),

(4, 'Patty Luna', 2, 78);

Figure

*Students Table*

The script below creates the courses table with the provided attributes and makes “courseId” the primary key.

CREATE TABLE `courses` (

`courseId` int(11) NOT NULL,

`courseName` varchar(45) NOT NULL

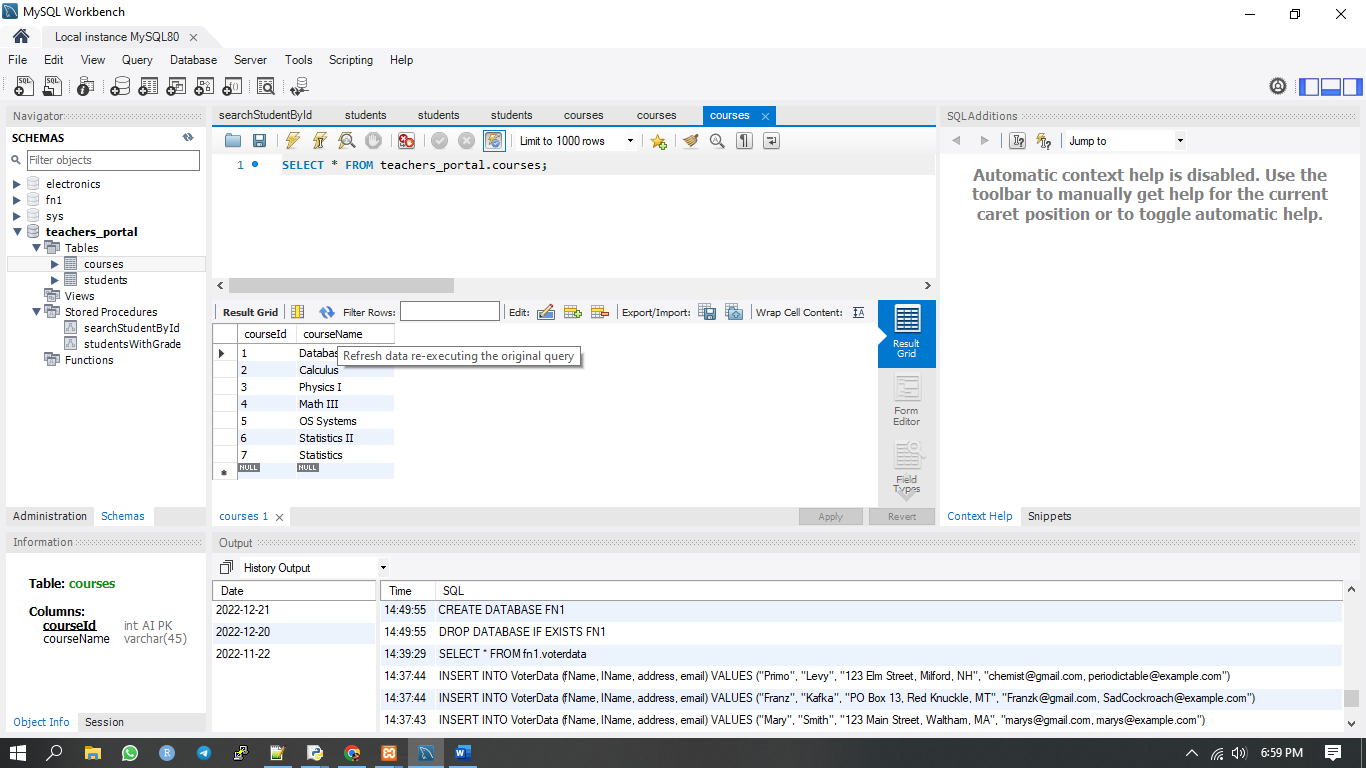
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

INSERT INTO `courses` (`courseId`, `courseName`) VALUES

(1, 'Database Design'),

(2, 'Calculus'),

(3, 'Physics I');

The second part of the script populates the table with the values provided.

Figure

*Course Table*

## Stored Procedure

The script below creates a stored procedure with the name “studentsWithGrade” that displays the home page table with students table LEFT JOINED to courses table. The procedure selects studentName, studentId, courseName, and grade columns from the two tables.

CREATE DEFINER=`root`@`localhost` PROCEDURE `studentsWithGrade` ()

BEGIN

SELECT students.studentName, students.studentId, courses.courseName, students.grade

FROM students LEFT JOIN courses ON students.enrolledInCourseID = courses.courseID;

END$$

# Python MySQL

## 1. Default Parameter Values

On my computer, MySQL server is running on port 3306, my username is “root”, and the root user password is “Firms@2215'”. Therefore, the updated parameter values on the portalDatabase.py file are as below:

def \_\_init\_\_(self,

host="localhost",

port="3307",

database="teachers\_portal",

user='root',

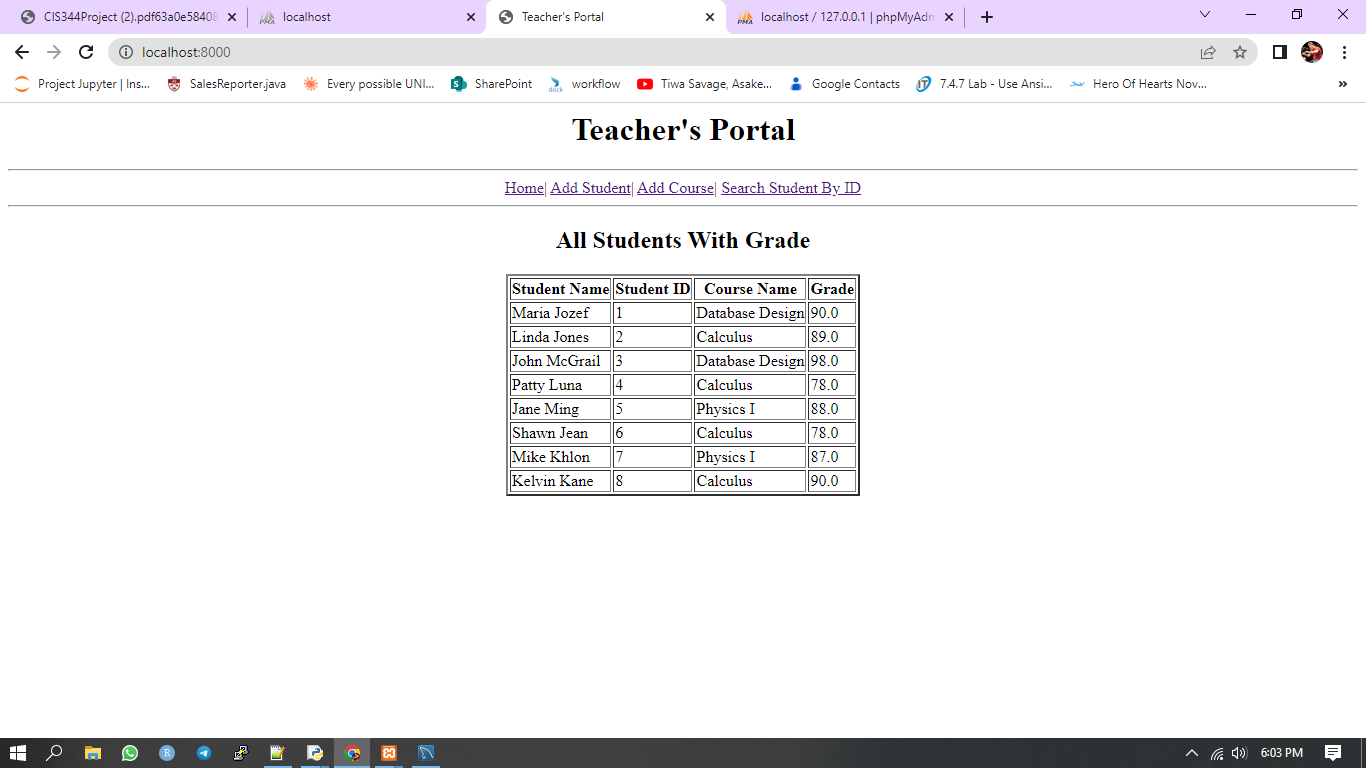
password='Firms@2215'):

## 2. Running portalServer.py

Successful connection allows the “portalServer.py” script to use the stored procedure “studentsWithGrade” to display the home page table on the web browser. The following is a screen shot of the results in a web browser after a successful connection.

Figure

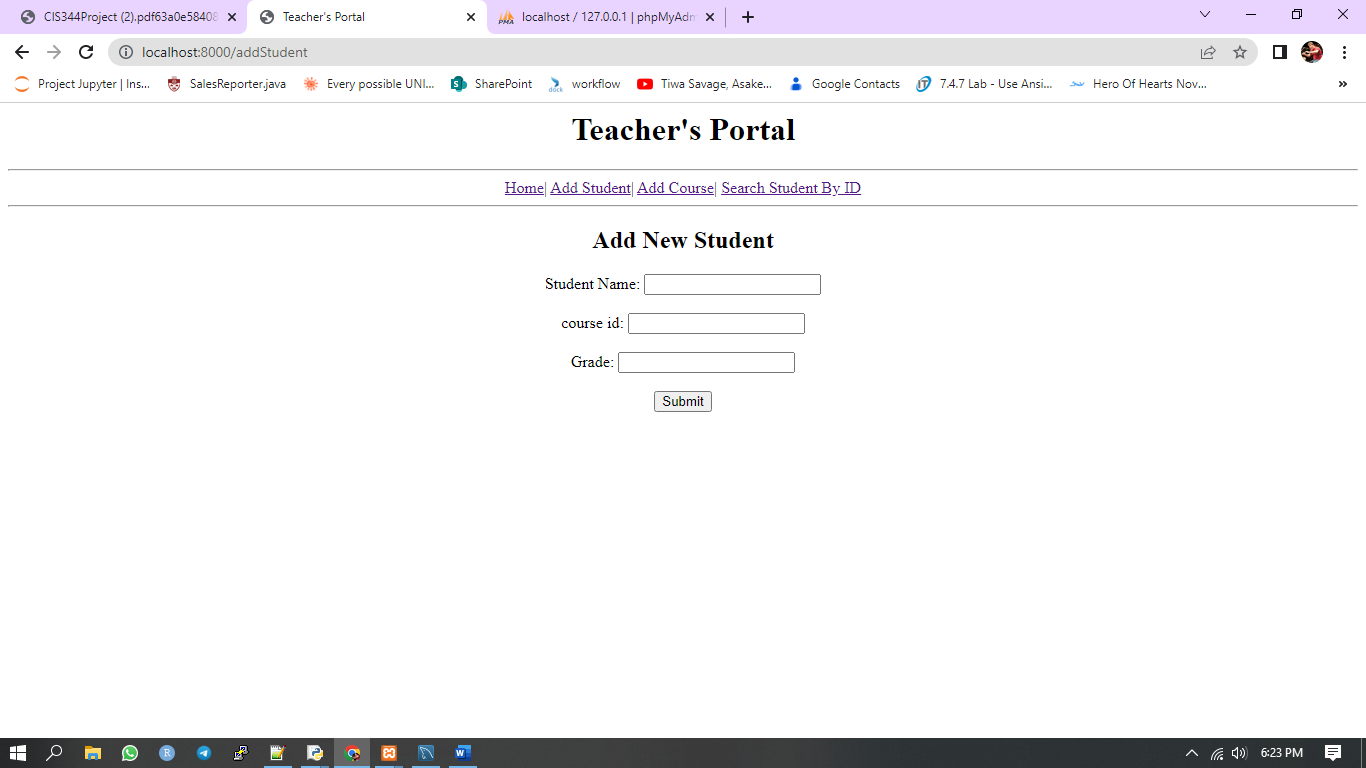
*Teacher’s Portal from Localhost Address*



## 3. Add Student Link

Figure

*Add New Student Form*



## 4. Add Student Function

To successfully add student into the database using the code file portalDatabase.py, I completed the method as below:

def addStudent(self, name, courseID, grade=0):

if self.connection.is\_connected():

self.cursor= self.connection.cursor();

add\_student = ("INSERT INTO students (studentName, enrolledInCourseID, grade)"

"VALUES (%s, %s, %s)")

record = (name, courseID, grade)

# add new student

self.cursor.execute(add\_student, record)

studentId = self.cursor.lastrowid

# commit to database

self.connection.commit()

# close cursor and connection

self.cursor.close()

self.connection.close()

pass

The query “INSERT INTO students (studentName, enrolledInCourseID, grade) VALUES (%s, %s, %s)” will receive input from the form and insert into MySQL database table. The execute function will execute the query and the commit function will post the data into the database table. The if self.connection.is\_connected()method checks if there is a connection to the database before the query is executed.

The code below calls the add student function and takes the form input as parameters in the portalServer.py code file. The table’s primary key is not part of the data input as in autoincrements.

## get form post values

student\_name = form.getvalue("sname")

student\_courseid = int(form.getvalue("courseid"))

student\_grade = float(form.getvalue("sgrade"))

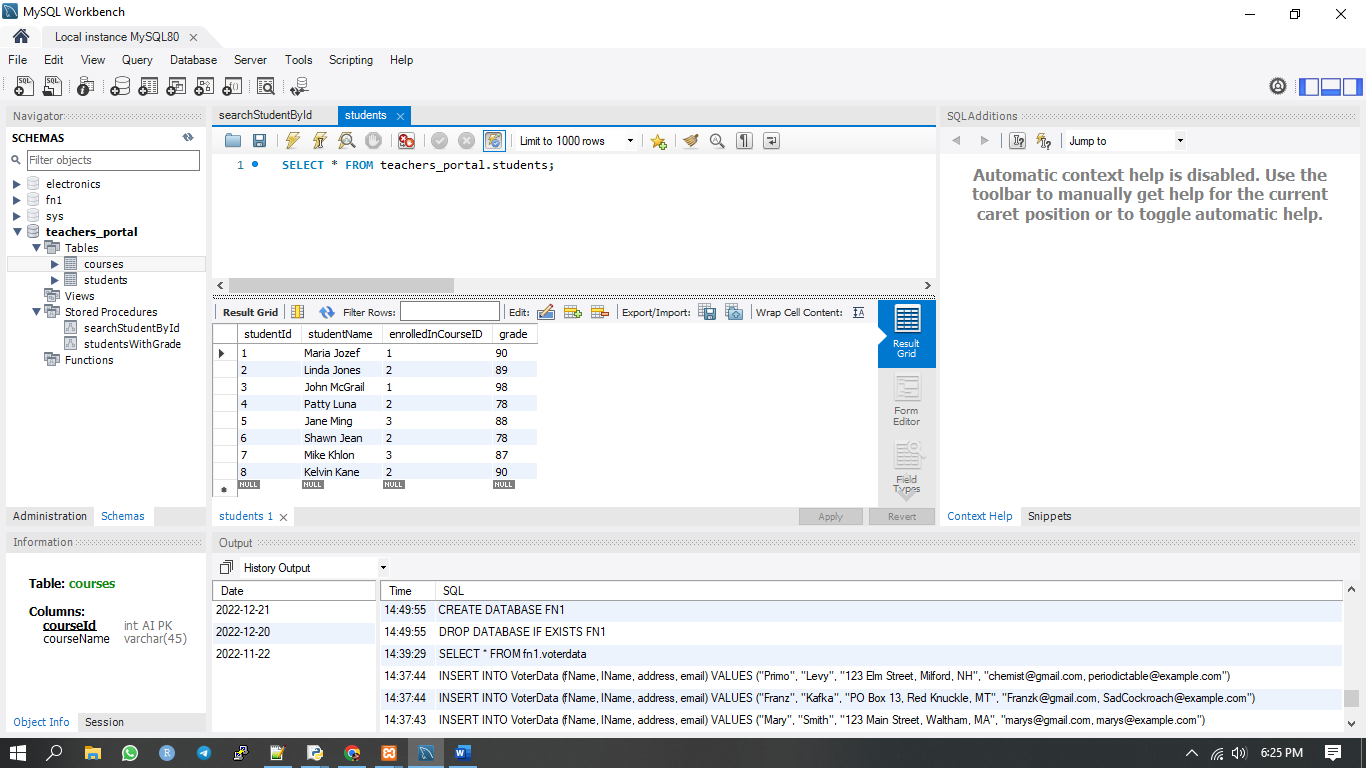
##Call the Database Method to a add a new student

self.database.addStudent(student\_name, student\_courseid, student\_grade)

The results are as presented in the figures below.

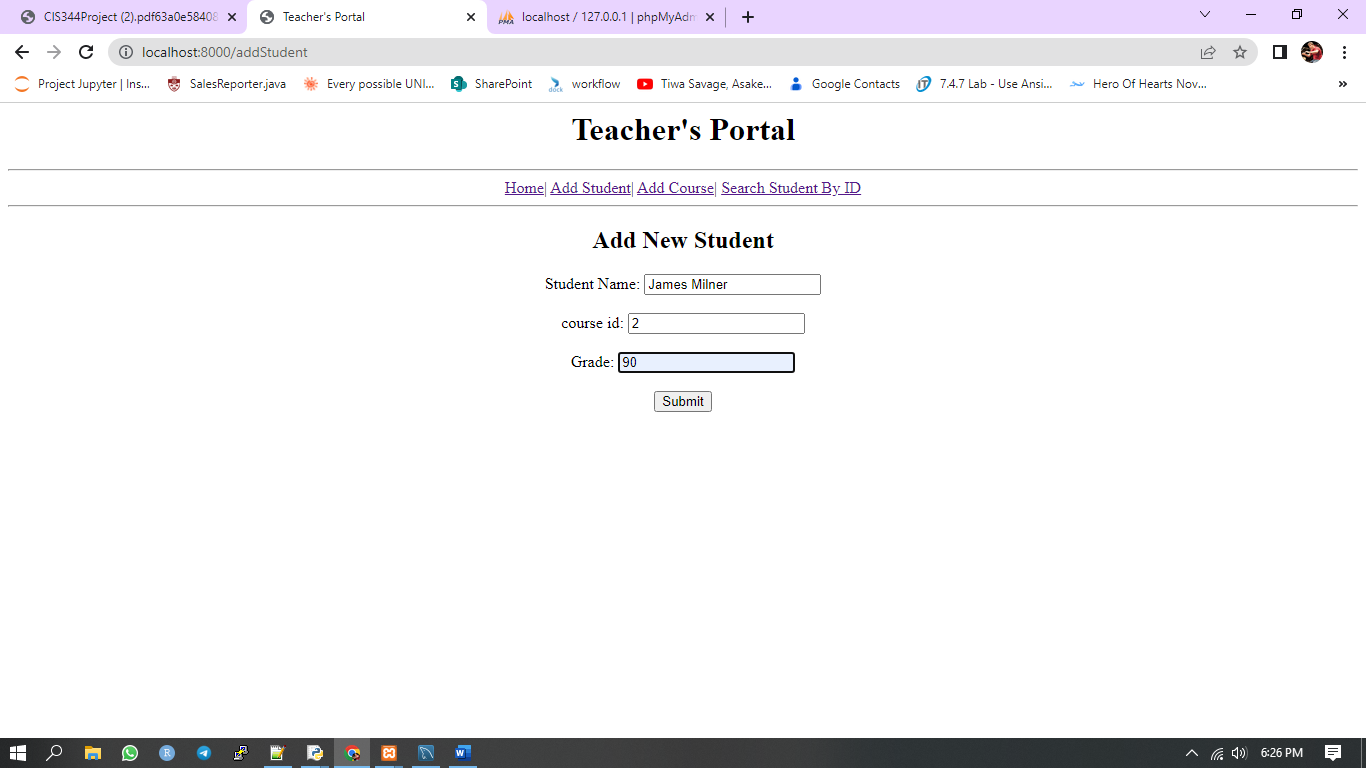
Figure

*Initial Rows of Data in Students Table*



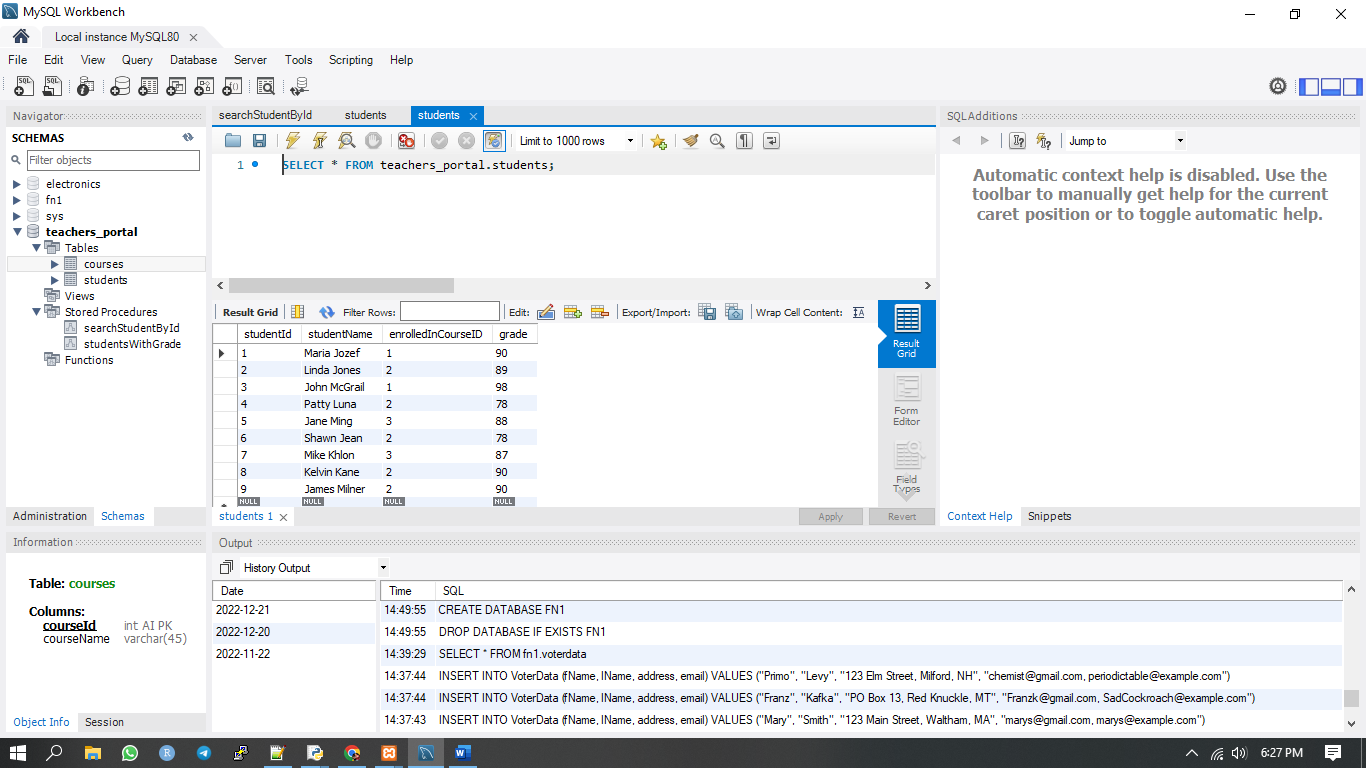
Figure

*Updating Students’ Record Using The Form From portalDatabase.Py*



Figure

*Updated Students Table*



# Extra Bonus

# 5. Extra Bonus: addCourse () Method in portalDatabase and portalServer

The addCourse () method was completed the same way as addStudent () method as I the script below.

def addCourse(self, name):

if self.connection.is\_connected():

self.cursor= self.connection.cursor();

add\_course = ("INSERT INTO courses (courseName)"

"VALUES (%s)")

record = (name)

# add new course

self.cursor.execute(add\_course, (record,))

courseId = self.cursor.lastrowid

# commit to database

self.connection.commit()

# close cursor and connection

self.cursor.close()

self.connection.close()

pass

The form’s input is taken as parameters for use in the portalServer.py code file.

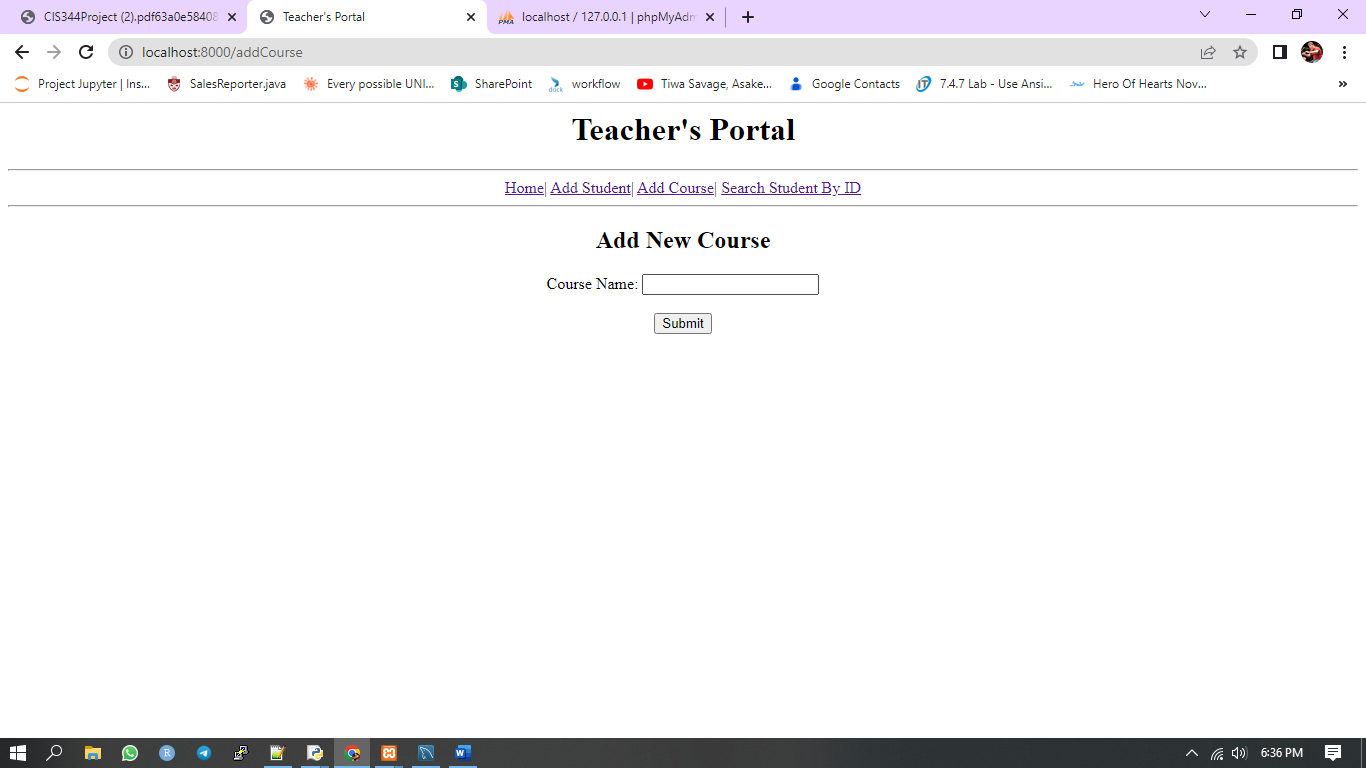
course\_name = form.getvalue("scourse")

## Call the Database Method to a add a new Course

self.database.addCourse(course\_name)

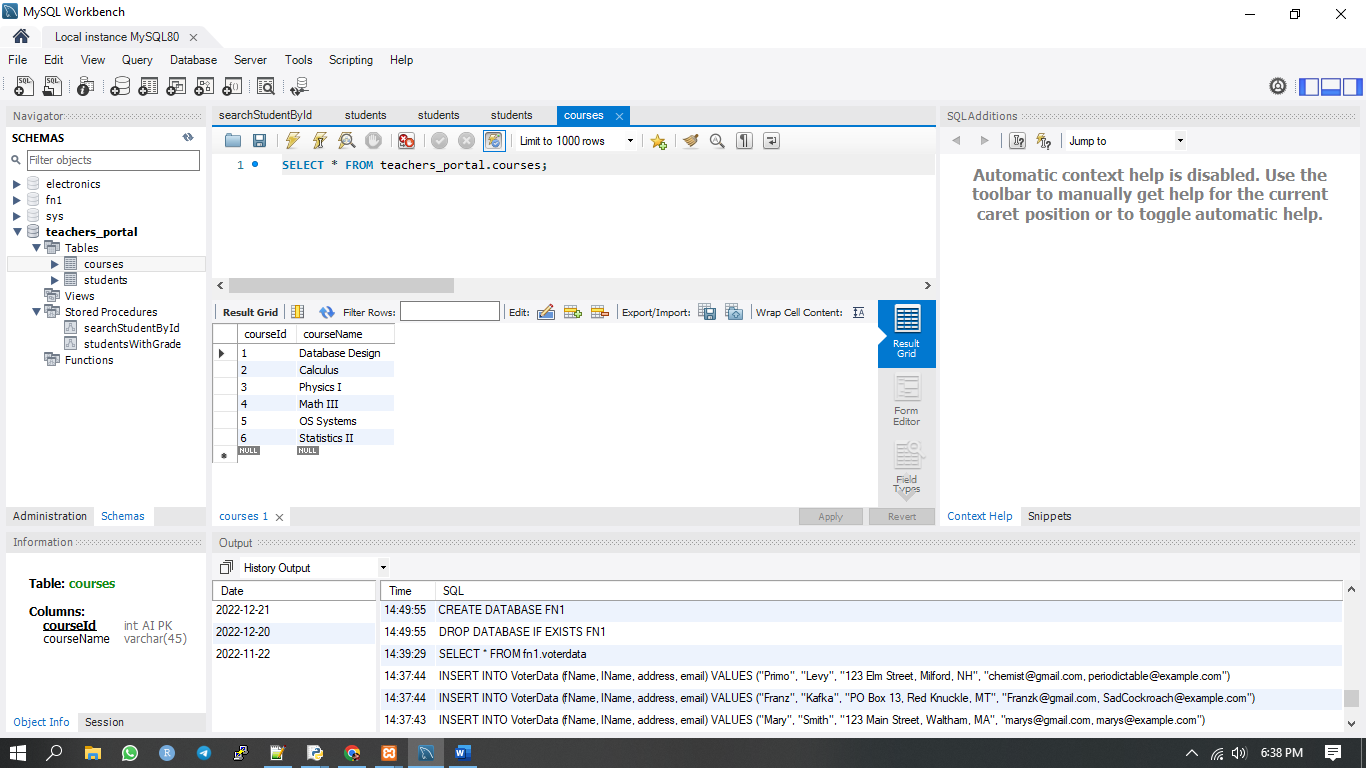
Figure

*Add Course Form*



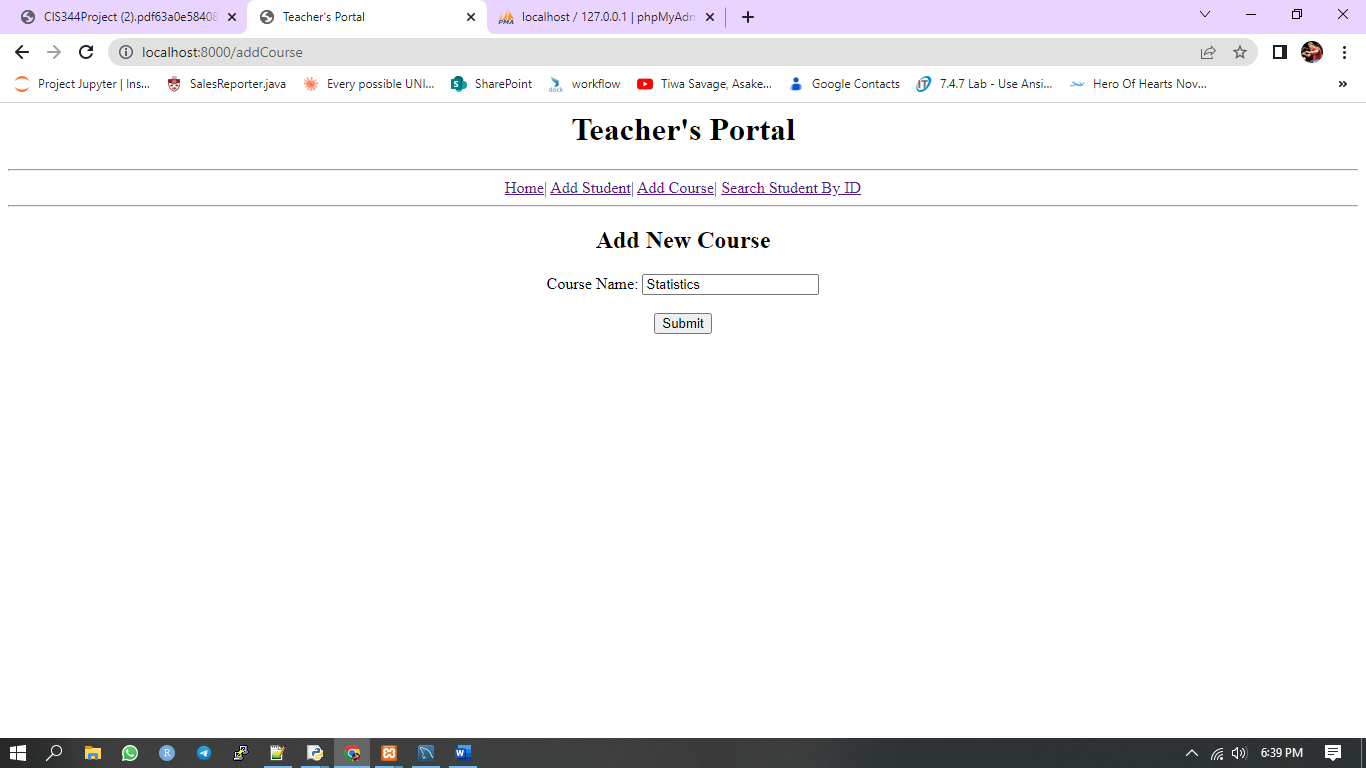
Figure

*Initial Rows of Data in The Courses Table.*



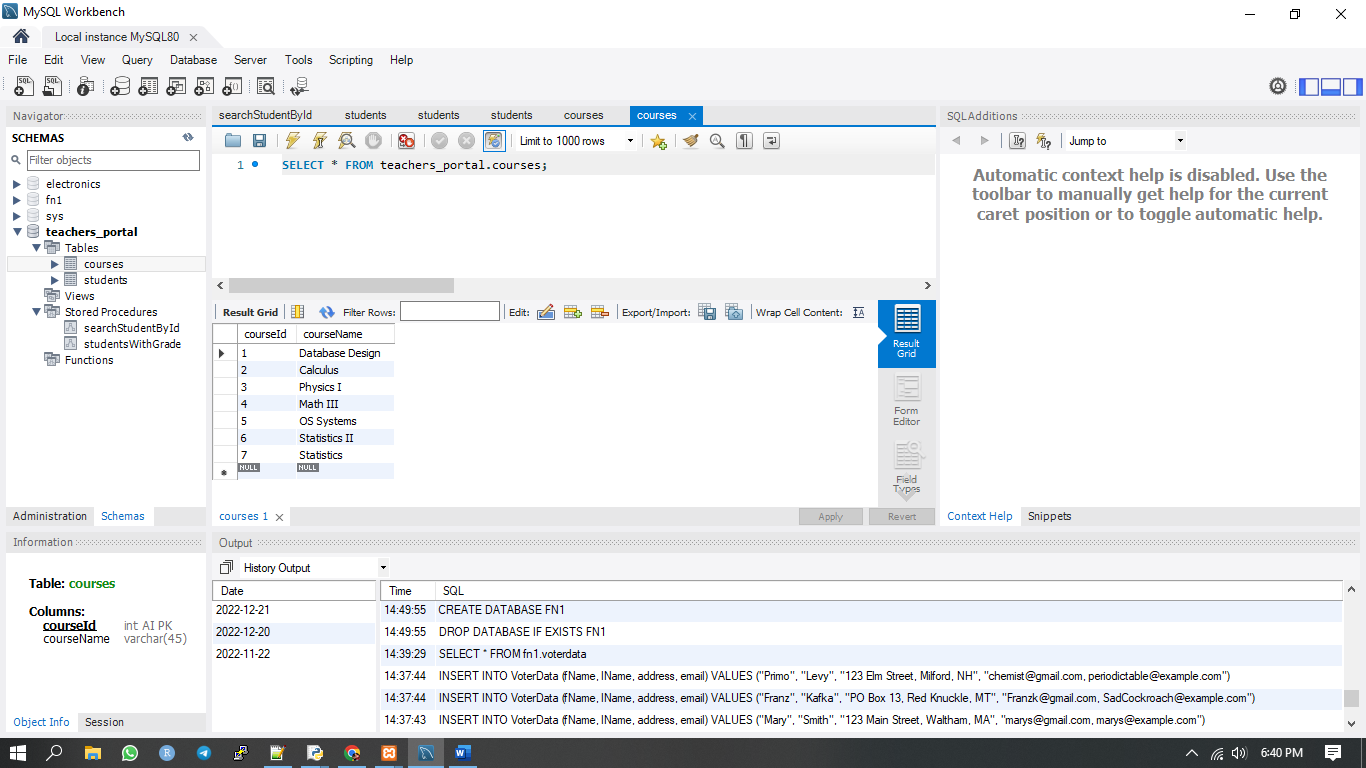
Figure

*Updating Courses’ Record Using the New Course Form on portalDatabase.py*



Figure

*Updated Courses Table*



# 6. Extra Bonus: searchStudent () in portalDatabase and portalServer

Implementing the searchStudent method involved creating a stored procedure to accept student ID as a parameter, as depicted below.

CREATE DEFINER=`root`@`localhost` PROCEDURE `searchStudentById` (IN `studId` INT)

BEGIN

SELECT \* FROM students WHERE studentId = studId;

END$$

The next step was to create a function in the portalDatabase program to call the stored procedure and accept the parameter student id.

def searchStudent(self, s\_id):

if self.connection.is\_connected():

self.cursor= self.connection.cursor();

self.cursor.callproc("searchStudentById", (s\_id,))

records = self.cursor.stored\_results()

return records

Finally, I created a form and called the function in the portalServer and stored the fetched data in a list. I used a for loop to print the data in a table on the HTTP server as shown below.

if self.path == '/searchStudent':

form = cgi.FieldStorage(

fp=self.rfile,

headers=self.headers,

environ={'REQUEST\_METHOD': 'POST'}

)

student\_id = int(form.getvalue("search"))

student\_data = []

records = self.database.searchStudent(student\_id)

for record in records:

student\_data=record.fetchall()

self.wfile.write(b"<html><head><title> Teacher's Portal </title></head>")

self.wfile.write(b"<body>")

self.wfile.write(b"<center><h1>Teacher's Portal</h1>")

self.wfile.write(b"<hr>")

self.wfile.write(b"<div> <a href='/'>Home</a>| \

<a href='/addStudent'>Add Student</a>|\

<a href='/addCourse'>Add Course</a>|\

<a href='/searchStudent'>Search Student By ID </a></div>")

self.wfile.write(b"<hr>")

self.wfile.write(b"<h3>Student Details</h3>")

self.wfile.write(b"<hr>")

self.wfile.write(b"<table border=2> \

<tr><th>Student ID </th>\

<th> Student Name </th>\

<th> Course ID </th>\

<th> Grade </th></tr>")

for row in student\_data:

self.wfile.write(b' <tr> <td>')

self.wfile.write(str(row[0]).encode())

self.wfile.write(b'</td><td>')

self.wfile.write(row[1].encode())

self.wfile.write(b'</td><td>')

self.wfile.write(str(row[2]).encode())

self.wfile.write(b'</td><td>')

self.wfile.write(str(row[3]).encode())

self.wfile.write(b'

</td></tr>')

self.wfile.write(b"<form action='/searchStudent' method='post'>")

self.wfile.write(b'<label for="search">Student ID:</label>\

<input type="number" id="search" name="search"><br><br>\

<input type="submit" value="Submit">\

</form>')

self.wfile.write(b"</table></center>")

self.wfile.write(b"</center></body></html>")

return

if self.path =='/searchStudent':

self.send\_response(200)

self.send\_header('Content-type','text/html')

self.end\_headers()

self.wfile.write(b"<html><head><title> Teacher's Portal </title></head>")

self.wfile.write(b"<body>")

self.wfile.write(b"<center><h1>Teacher's Portal</h1>")

self.wfile.write(b"<hr>")

self.wfile.write(b"<div> <a href='/'>Home</a>| \

<a href='/addStudent'>Add Student</a>|\

<a href='/addCourse'>Add Course</a>|\

<a href='/searchStudent'>Search Student By ID </a></div>")

self.wfile.write(b"<hr><h2>Search Student</h2>")

self.wfile.write(b"<form action='/searchStudent' method='post'>")

self.wfile.write(b'<label for="search">Student ID:</label>\

<input type="number" id="search" name="search"><br><br>\

<input type="submit" value="Submit">\

</form>')

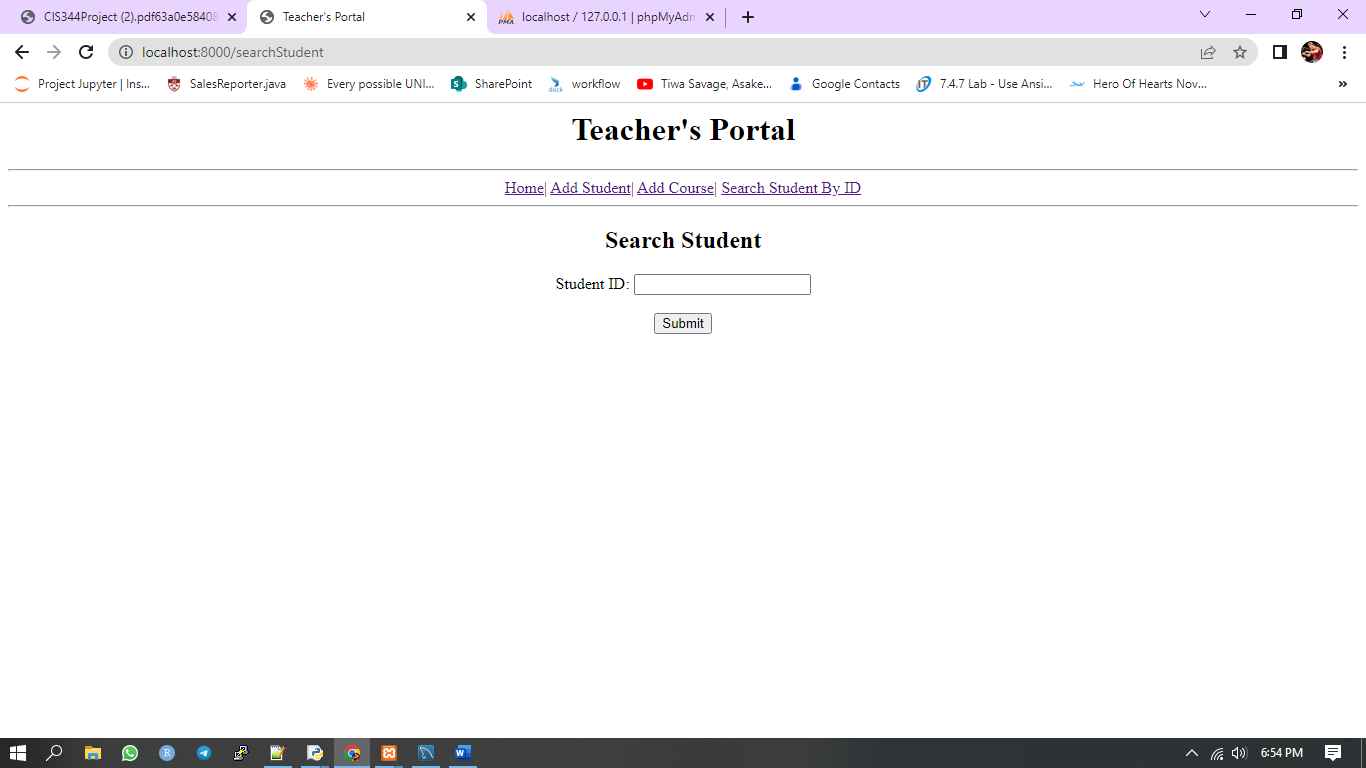
self.wfile.write(b"</center></body></html>")

return

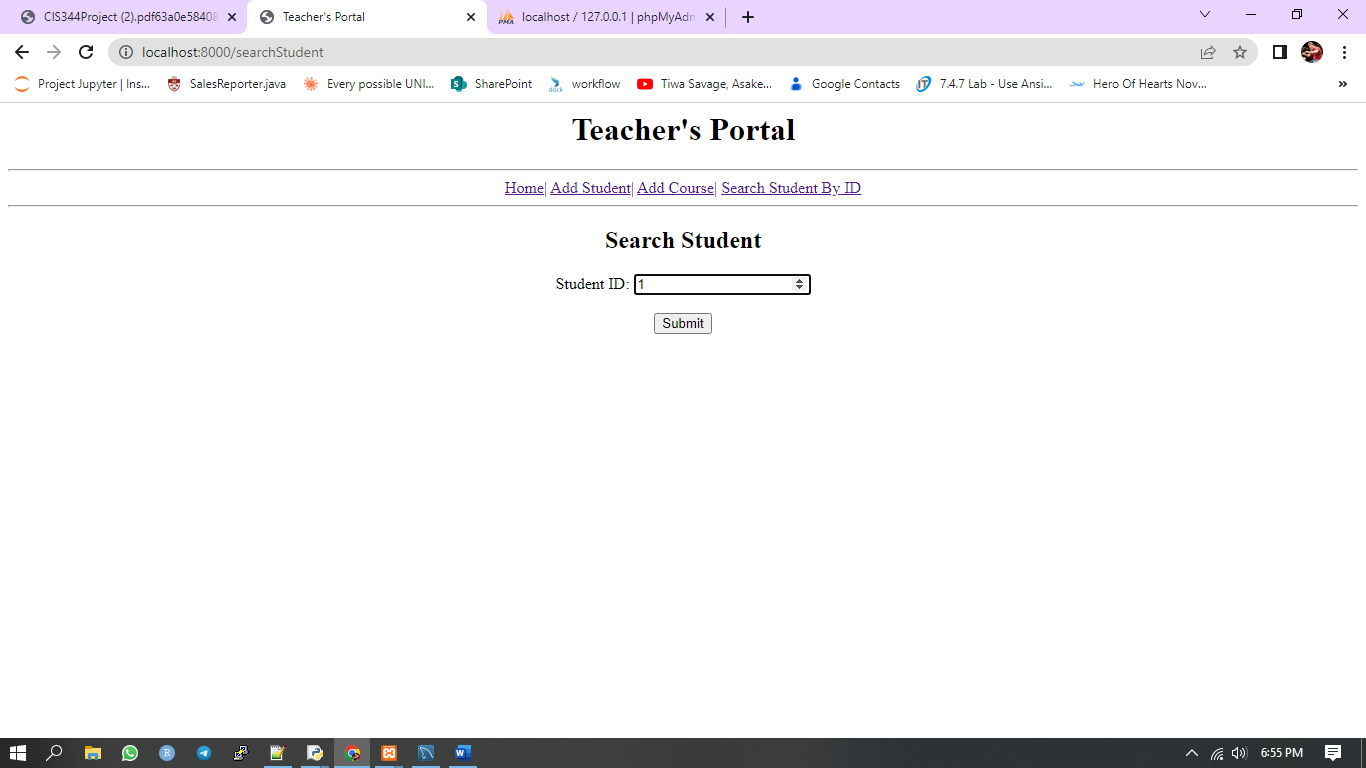
The results of the completed searchStudent () method are shown below.

Figure

*Search Student By ID Form*

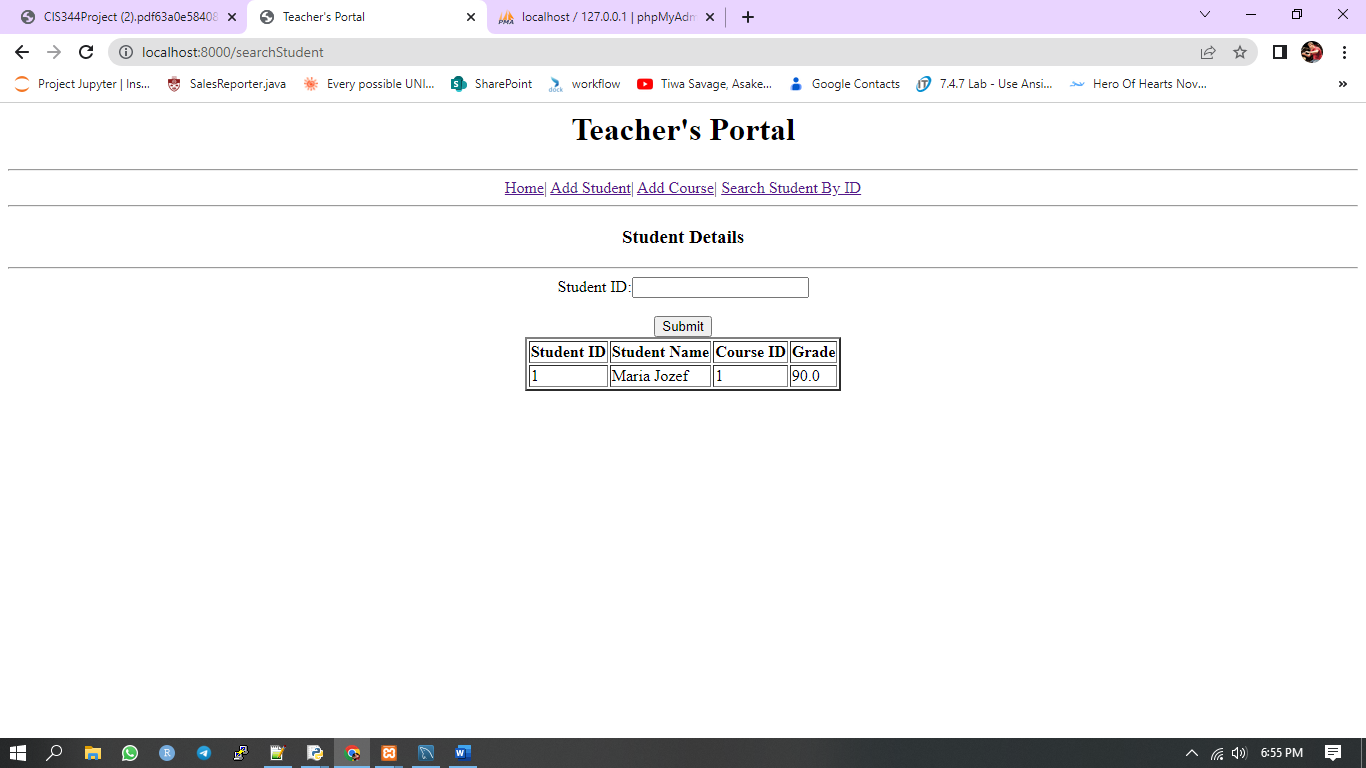


Figure

*Student Search*

Figure

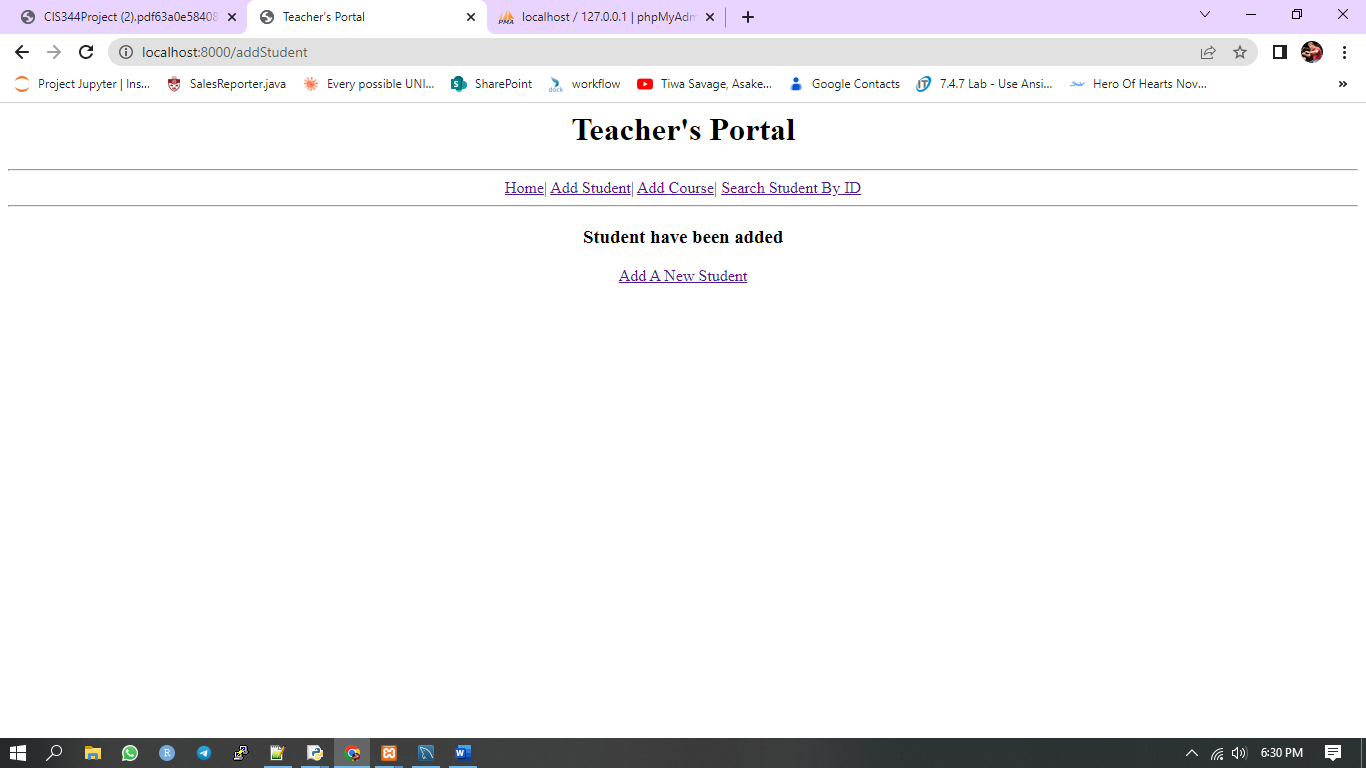
*Search Results*



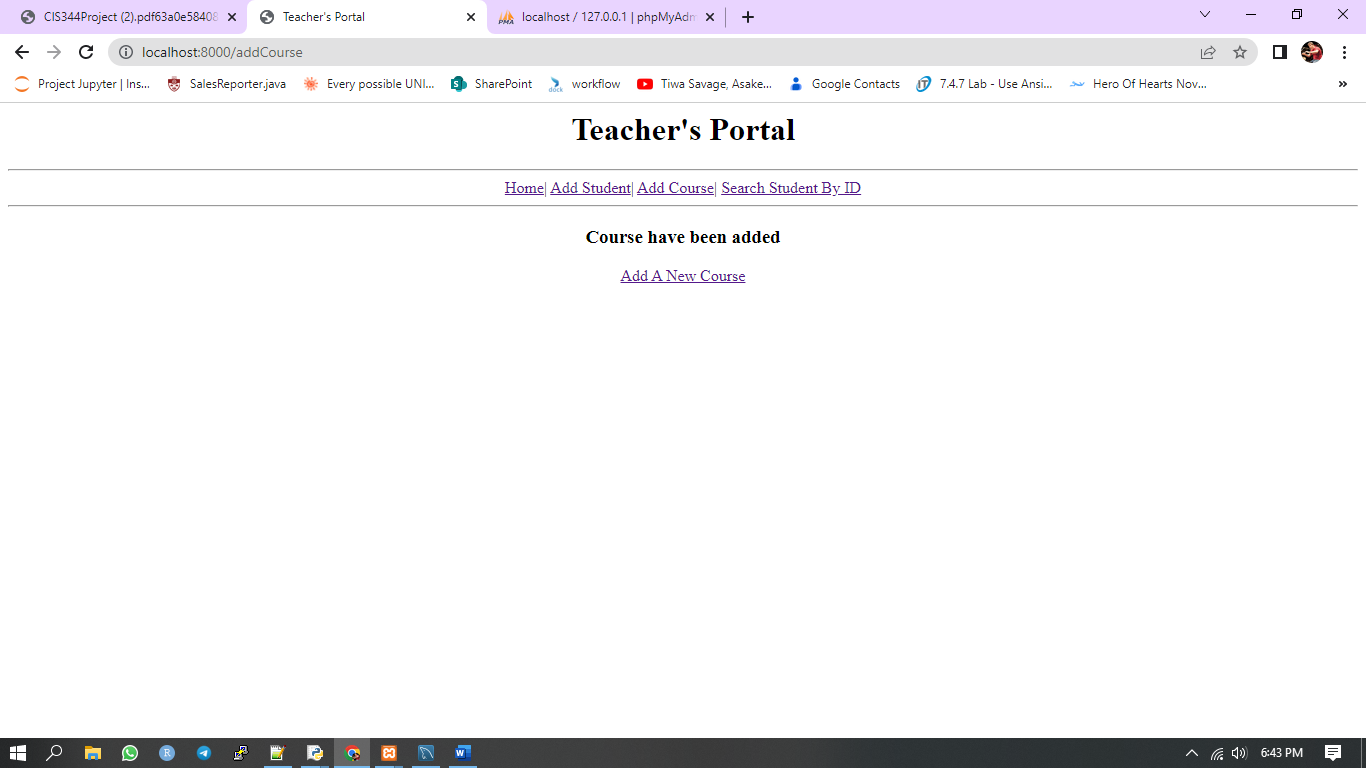
The source code and script file with MySQL queries were uploaded to GitHub in a public repository and are accessible through this link.

# Appendices

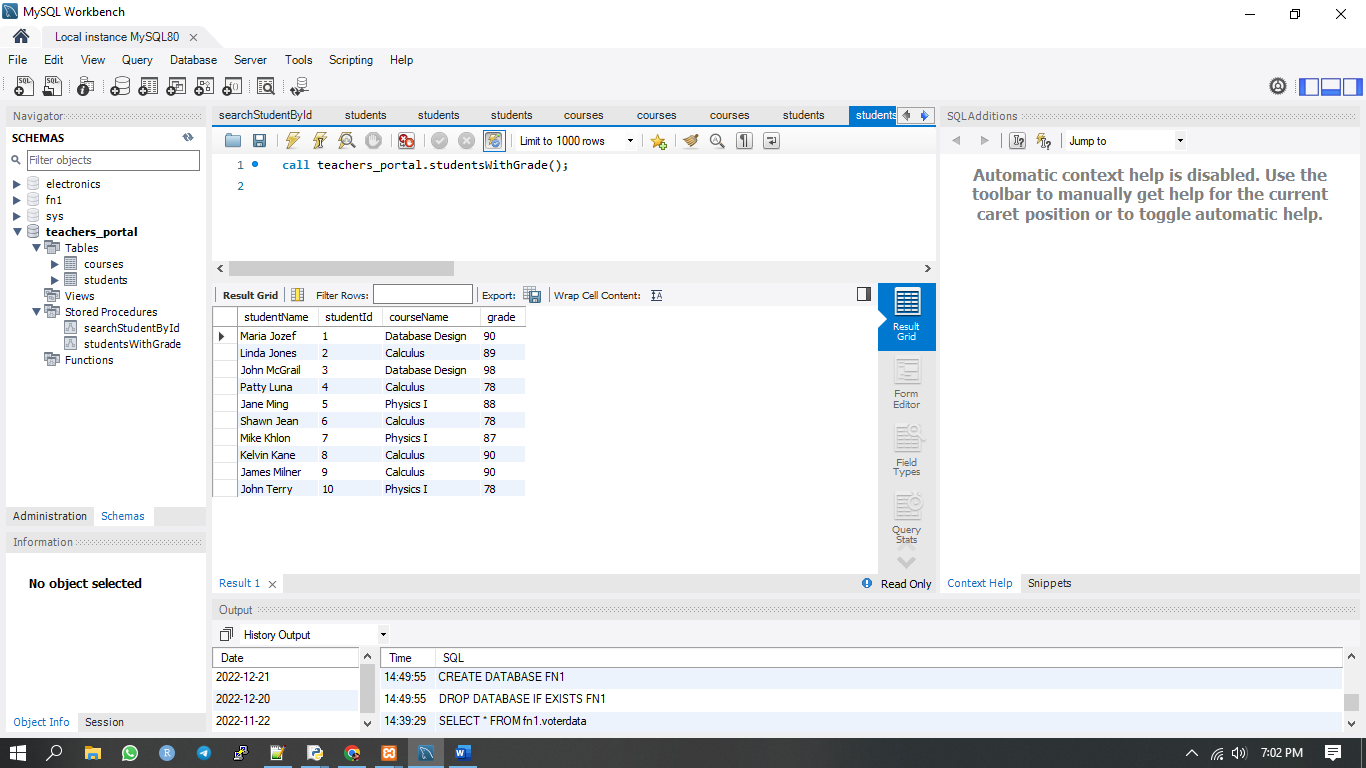
## Appendix 1: Confirmation Upon Adding a New Student

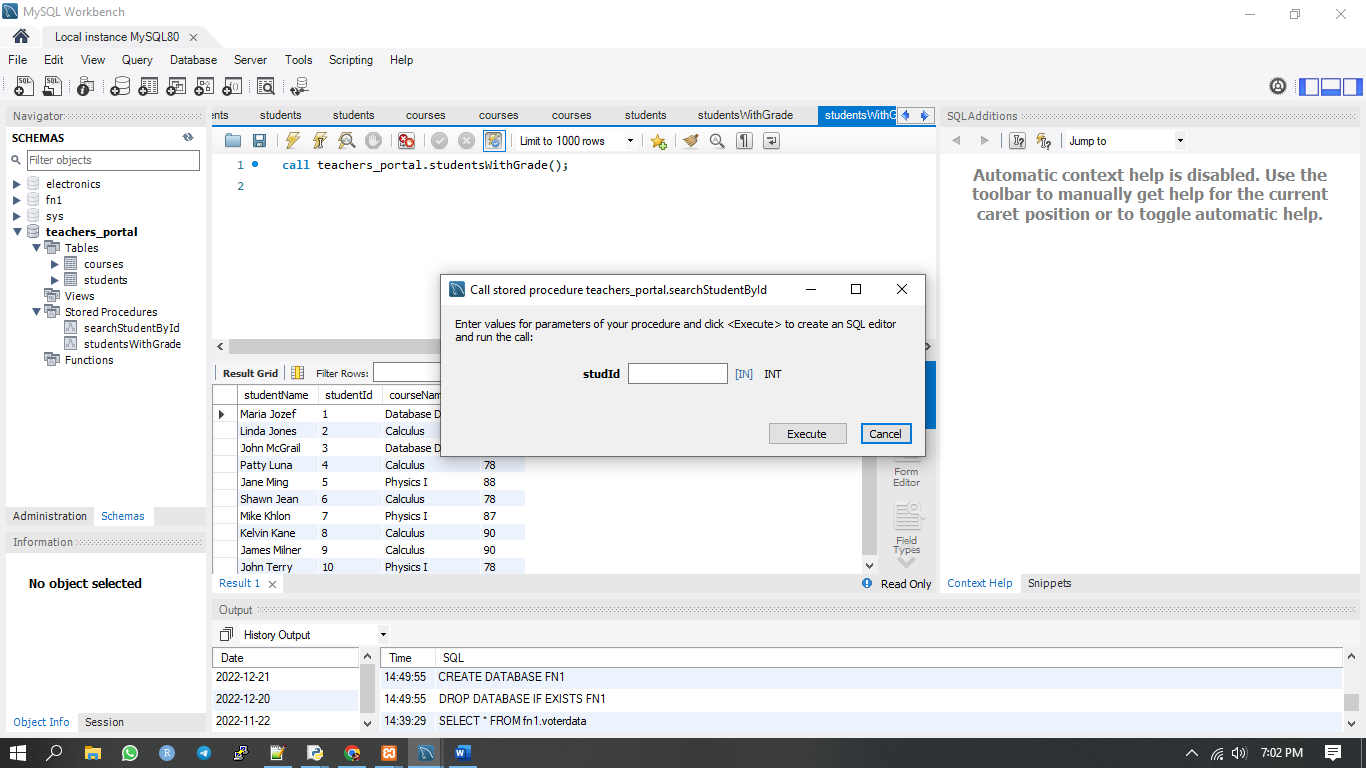


## Appendix 2: Confirmation Upon Adding a New Course



## Appendix 3: List of Students with Grades from The Students Table as Seen from MySQL Workbench





GitHub Repository Link

<https://github.com/Cypher356/Richmond-Ankomah.git>