**Assignment -2**

Python Programming

|  |  |
| --- | --- |
| Assignment Date | 29 September 2022 |
| Student Name | Rajarajeswari |
| Role | Team Member 2 |
| Student Roll Number | 92172019102104 |
| Maximum Marks | 2 Marks |
| Team ID | PNT2022TMID16927 |

**Question-1:**

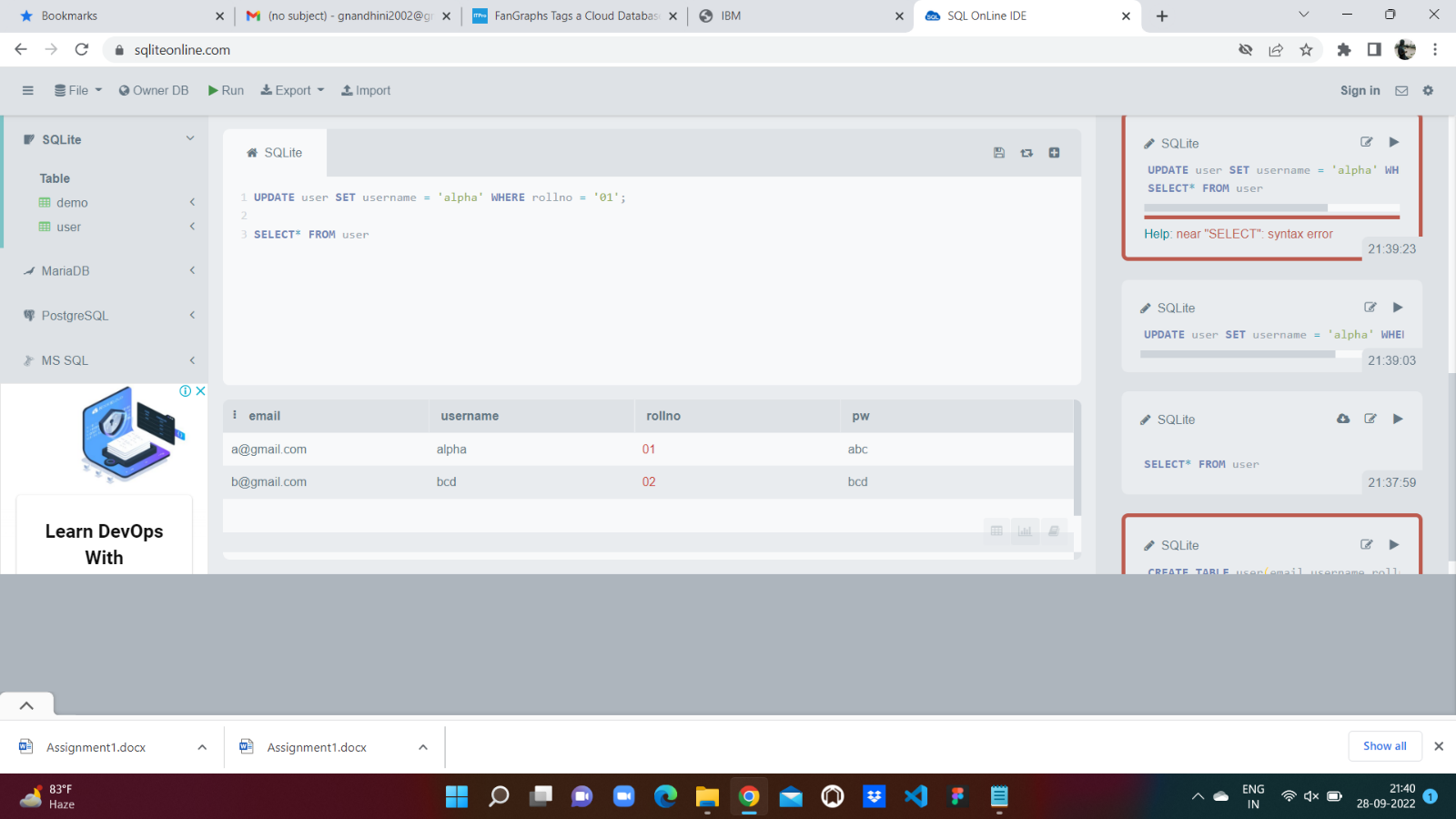
Create User table with user with email,username,roll number, password.

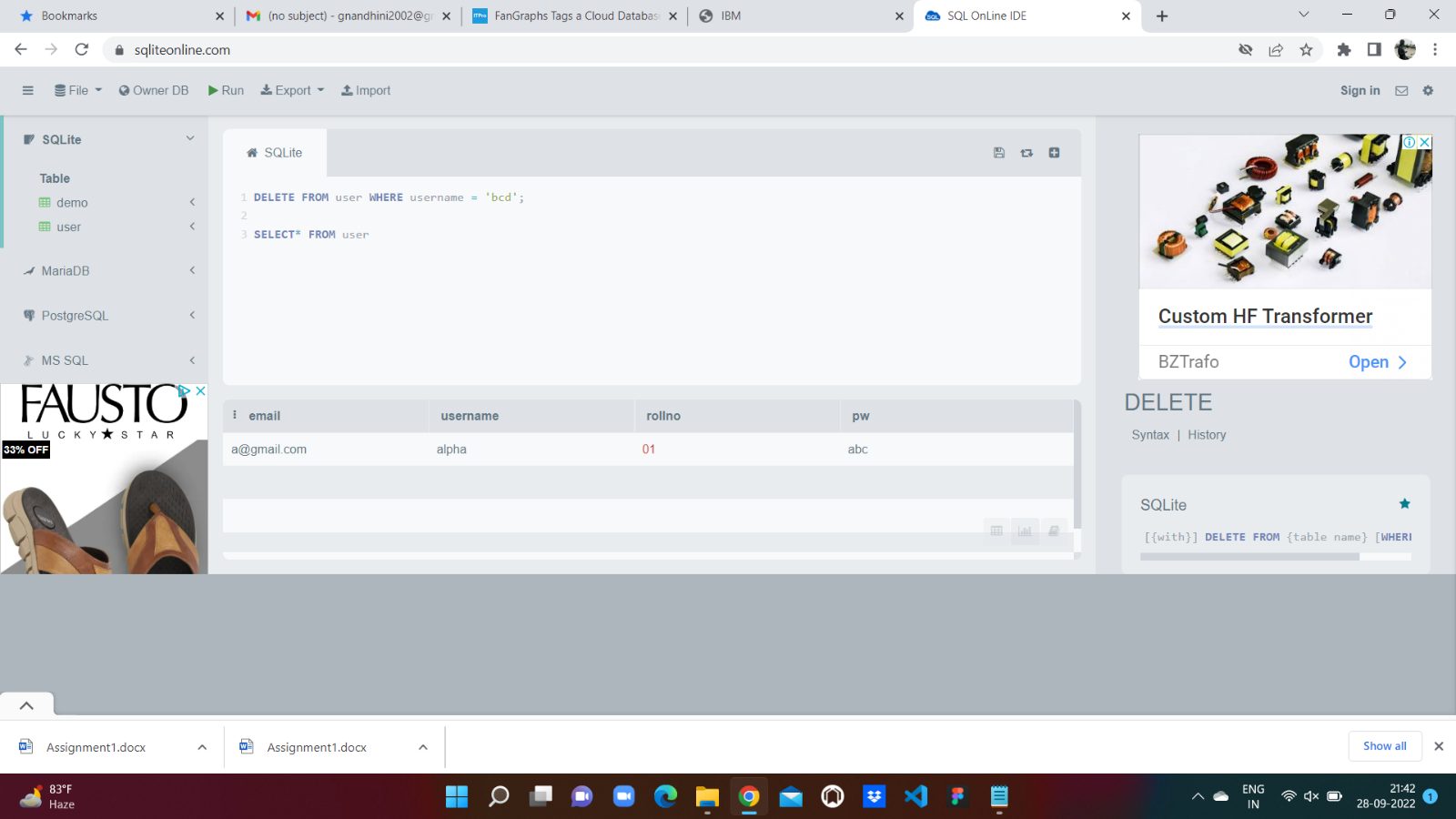
|  |
| --- |
| **Solution:**  CREATE TABLE user(email,username,rollno,pw);  INSERT INTO user (email,username,rollno,pw) VALUES ('a@gmail.com', 'abc','01','abc');  INSERT INTO user (email,username,rollno,pw) VALUES ('b@gmail.com', 'bcd','02','bcd');  INSERT INTO user (email,username,rollno,pw) VALUES ('c@gmail.com', 'cde','03','cde');  INSERT INTO user (email,username,rollno,pw) VALUES ('d@gmail.com', 'def','04','def'); |
|  |  |

**Question-2:**

Perform UPDATE,DELETE Queries with user table

|  |
| --- |
| **Solution:**  UPDATE user SET username = 'alpha' WHERE rollno = '01';  SELECT\* FROM user  DELETE FROM user WHERE username = 'bcd';  SELECT\* FROM user |





**Question-3:**

Connect python code to db2.

|  |
| --- |
| **Solution:**  pip install ibm\_db  from ibm\_db import connect  # Careful with the punctuation here - we have 3 arguments.  # The first is a big string with semicolons in it.  # (Strings separated by only whitespace, newlines included,  # are automatically joined together, in case you didn't know.)  # The last two are emptry strings.  connection = connect('DATABASE=<database name>;'  'HOSTNAME=<database ip>;' # 127.0.0.1 or localhost works if it's local  'PORT=<database port>;'  'PROTOCOL=TCPIP;'  'UID=<database username>;'  'PWD=<username password>;', '', '')  def results(command):  from ibm\_db import fetch\_assoc  ret = []  result = fetch\_assoc(command)  while result:  # This builds a list in memory. Theoretically, if there's a lot of rows,  # we could run out of memory. In practice, I've never had that happen.  # If it's ever a problem, you could use  # yield result  # Then this function would become a generator. You lose the ability to access  # results by index or slice them or whatever, but you retain  # the ability to iterate on them.  ret.append(result)  result = fetch\_assoc(command)  return ret # Ditch this line if you choose to use a generator.  from ibm\_db import tables  t = results(tables(connection))  from ibm\_db import exec\_immediate  sql = 'LIST \* FROM ' + t[170]['TABLE\_NAME']  rows = results(exec\_immediate(connection, sql)) |

**Question-4:**

Create a flask app with registration page, login page and welcome page. By default load the registration page once the user enters all the fields store the data in database and navigate to login page authenticate user username and password. If the user is valid show the welcome page

|  |
| --- |
| **Solution:**  @app.route('/login', methods=['GET', 'POST'])  def login():  error = None  if request.method == 'POST':  if request.form['username'] != 'admin' or request.form['password'] != 'admin':  error = 'Invalid Credentials. Please try again.'  else:  return redirect(url\_for('home'))  return render\_template('login.html', error=error)  **html code:**  <html>  <head>  <title>Flask Intro - login page</title>  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <link href="static/bootstrap.min.css" rel="stylesheet" media="screen">  </head>  <body>  <div class="container">  <h1>Please login</h1>  <br>  <form action="" method="post">  <input type="text" placeholder="Username" name="username" value="{{  request.form.username }}">  <input type="password" placeholder="Password" name="password" value="{{  request.form.password }}">  <input class="btn btn-default" type="submit" value="Login">  </form>  {% if error %}  <p class="error"><strong>Error:</strong> {{ error }}  {% endif %}  </div>  </body>  </html> |