# MODULE-7: Azure Cloud Exercises

## Exercise 1: Create and Configure a Virtual Machine

**Objective**: Create and configure Ubuntu and Windows Virtual Machines on Azure Portal.

- Create an Ubuntu VM: Log in to the Azure Portal. >> Navigate to Virtual Machines >> Create >> Choose Ubuntu Server 20.04 LTS >> Configure:
    - Size: Standard_B1s (or similar)
    - Authentication Type: SSH (generate a key pair if not available).
    - Inbound Port: Allow SSH (port 22).

- Create a Windows VM: Follow similar steps, selecting Windows Server 2022 >> Configure:
    - Size: Standard_B1s (or similar)
    - Authentication Type: SSH (generate a key pair if not available).
    - Inbound Port: Allow SSH (port 22).

- The Deployment is verified by accessing the default web page.

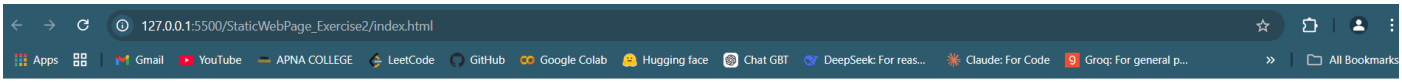
## Exercise 2: Deploy a Static Web Application

**Objective**: Host a static website using Azure App Service.

- Navigate to App Services >> Create:
    - Runtime Stack: Python 3.10 (or latest).
    - Operating System: Linux.
    - Region: Closest to your location.

- Deploy the application >> Upload a simple static website (e.g., index.html and CSS files) using FTP or the Kudu console.

- The Deployment is verified by accessing the default web page.




HelloWorld to Static WebSite Deployed on Azure

# Exercise 3: Deploy a Flask Application (Dynamic Web App)

**Objective**: Deploy a Python Flask application using Azure App Service.

- Create a Flask app:

M7_AzureCloud_Assignments > FlaskPythonApp > app.py > ...

You, last week | 1 author (You)

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def helloWorld():
    return "Hello World!"

@app.route('/message')
def message():
    return "This is a message page..."

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=8000)
```

You, last week • host and port

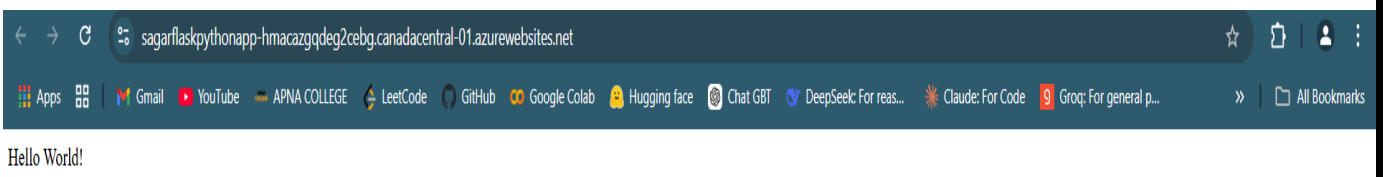
.

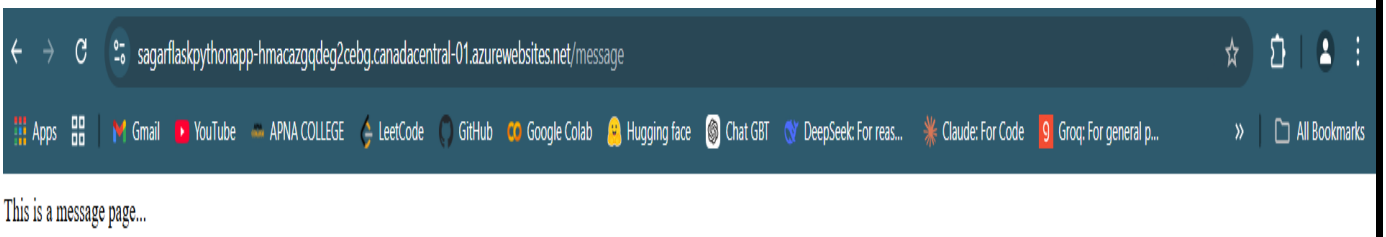
- Push the code to a GitHub repository.

- In the Azure Portal, navigate to App Services > Create. >> Configure:

  - Runtime Stack: Python 3.10 (or latest).
  - Deployment Source: Connect your GitHub repository.

- The Deployment is verified by accessing the default web page.



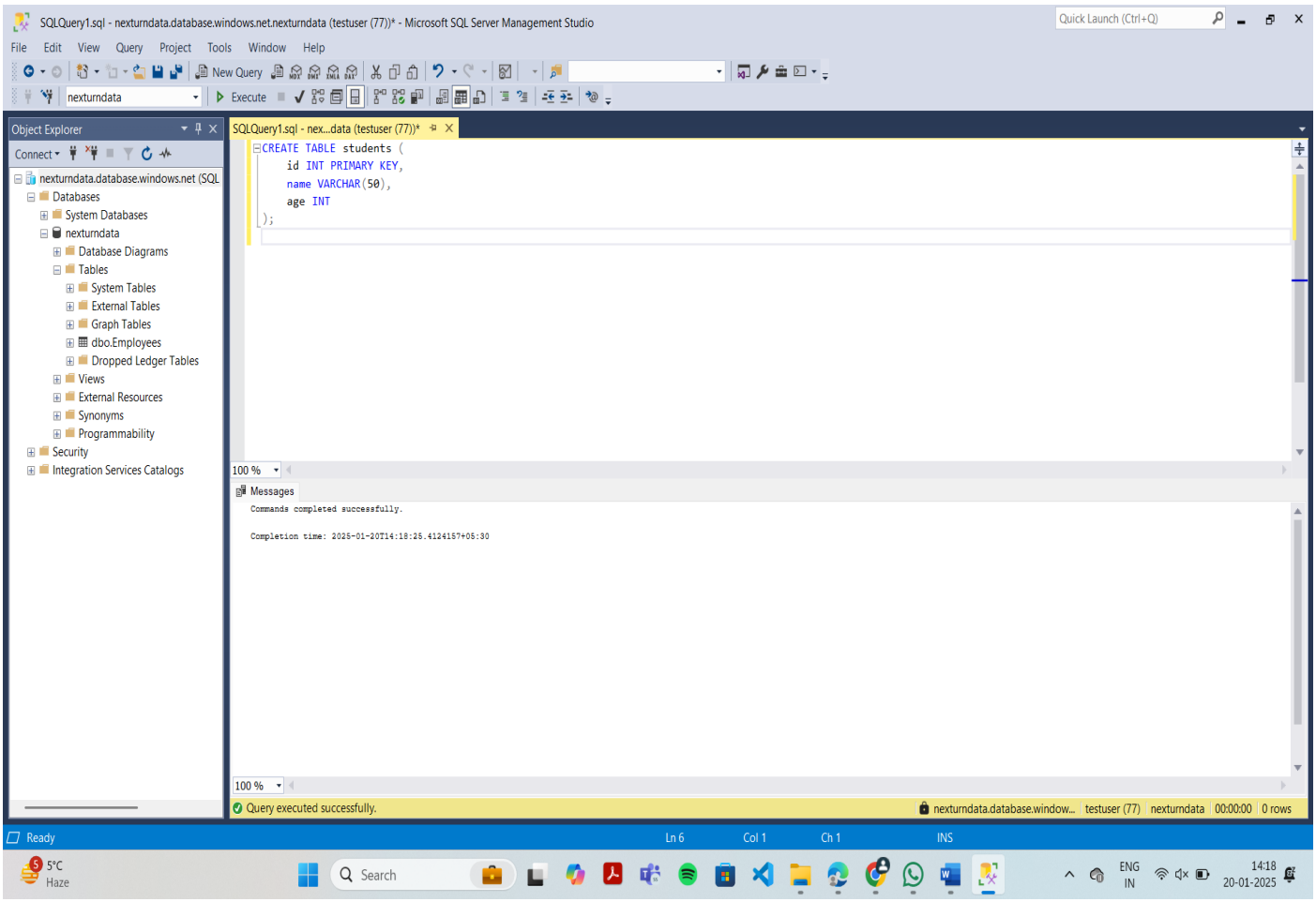
# Exercise 4: Set Up and Use an Azure SQL Database

**Objective**: Create an Azure SQL Database and connect to it from your local machine.

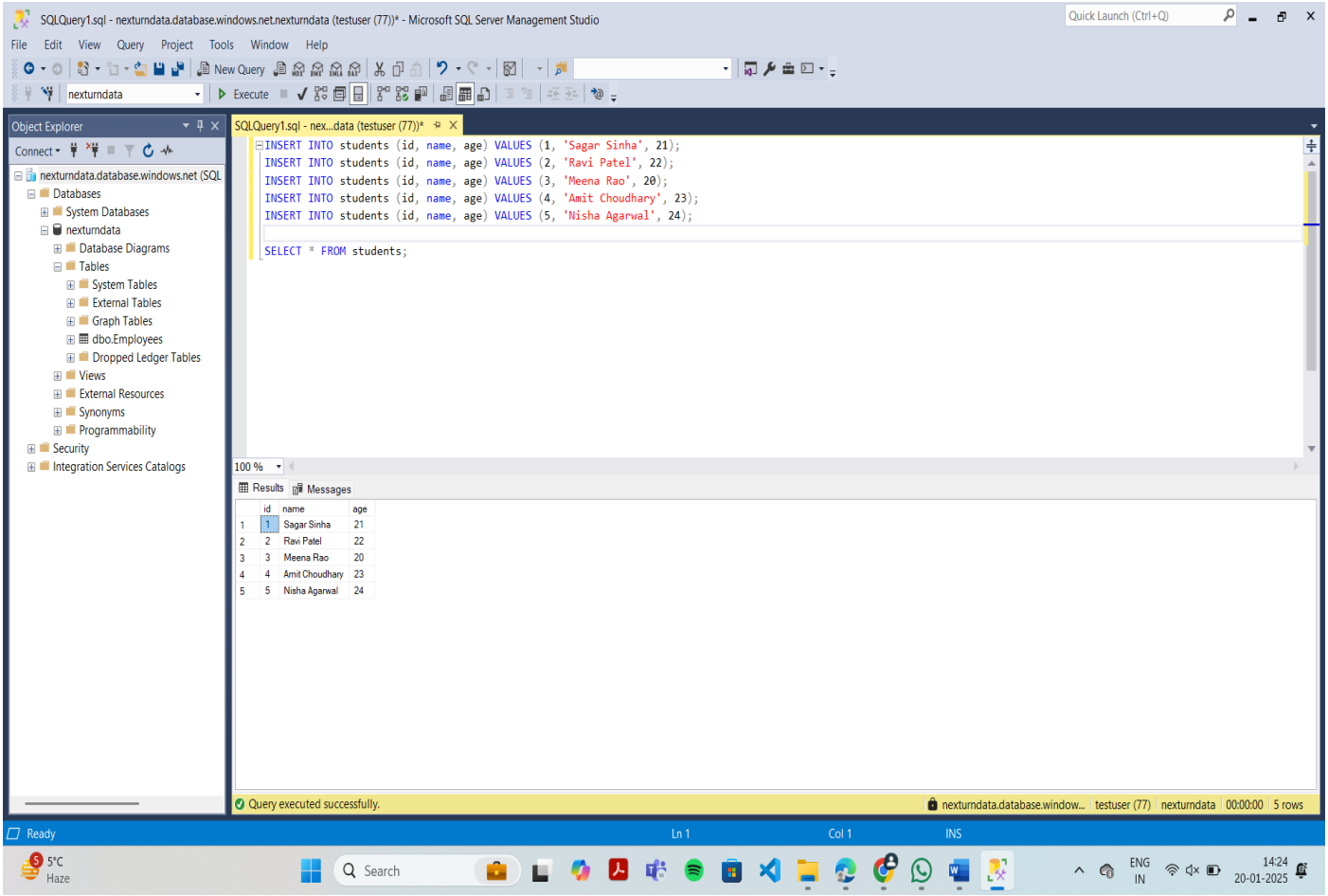
- Navigate to SQL Databases >> Create >> Config:
  - Database Name: StudentDB.
  - Server: Create a new server with username and password.
  - Compute + Storage: Use the free tier. (or latest).

- Deploy the database. >> Connect using Azure Data Studio or SQL Server Management Studio (SSMS).

- Task:
  - Create a table Students with columns ID, Name, and Age.




- Insert sample data and query it.




# Exercise 5: Integrate Flask App with Azure SQL Database

**Objective**: Connect a Flask app to Azure SQL Database and perform CRUD operations.

- Navigate to Use the Flask app from Exercise 3. >> Install required libraries:
  - pip install flask pyodbc

- Modify the app to connect to the SQL Database:

```
import pyodbc

# Database connection
conn_str = (
    'DRIVER={ODBC Driver 17 for SQL Server};'
    'SERVER=nexturndata.database.windows.net;'
    'DATABASE=nexturndata;'
    'UID=testuser;'
    'PWD=Sagar@123456789'
)
```

- Add a route to fetch and display data from the Students table >> Deploy the updated app to Azure App Service.

```python
# CRUD Operations
@app.route('/students', methods=['GET'])
def get_students():
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM Students')
    students = []
    for row in cursor.fetchall():
        students.append({
            'id': row[0],
            'name': row[1],
            'age': row[2]
        })
    conn.close()
    return jsonify(students)
```

- The CURD functionality is verified by using POSTMAN.

# ---------END---------