ARQUITECTURA Y CÓDIGO DE LA SOLUCIÓN

La solución implementada sigue una arquitectura de tipo cliente-servidor en Python con las siguientes características clave.

Interfaz Gráfica: Se utiliza Tkinter para crear una interfaz gráfica simple que permite al usuario ingresar las credenciales de Azure Blob Storage y la URL de conexión de PostgreSQL. La interfaz también incluye un botón para ejecutar la actualización de los datos desde Azure Blob Storage, una imagen de fondo llamada Definit.png y un indicador que muestra la última hora de actualización.

Código de Interfaz Gráfica en donde se aprecia detalladamente la arquitectura de esta pequeña interfaz a falta de tiempo quedo pendiente la simulación del logo de carga de periferia en formato gif.

```
    root = tk.Tk()
    root.title("Actualizador de Datos")

    image_path = os.path.join(os.path.dirname(__file__), 'imagen', 'Definit.png')
    bg_image = Image.open(image_path)
    bg_photo = ImageTk.PhotoImage(bg_image)

    root.geometry(f"{bg_image.width}x{bg_image.height}")

    bg_label = tk.Label(root, image=bg_photo)
    bg_label.place(relwidth=1, relheight=1)

    azure_frame = tk.Frame(root, bg='white', bd=5)
    azure_frame.place(relx=0.5, rely=0.20, relwidth=0.75, relheight=0.25, anchor='n')

    tk.Label(azure_frame, text="Azure Account URL:").grid(row=0, column=0, sticky='w')
    azure_account_entry = tk.Entry(azure_frame)
    azure_account_entry.grid(row=0, column=1)

    tk.Label(azure_frame, text="Container Name:").grid(row=1, column=0, sticky='w')
    azure_container_entry = tk.Entry(azure_frame)
    azure_container_entry.grid(row=1, column=1)

    tk.Label(azure_frame, text="SAS Token:").grid(row=2, column=0, sticky='w')
    azure_sas_token_entry = tk.Entry(azure_frame)
    azure_sas_token_entry.grid(row=2, column=1)

    postgres_frame = tk.Frame(root, bg='white', bd=5)
    postgres_frame.place(relx=0.5, rely=0.45, relwidth=0.75, relheight=0.15, anchor='n')

    tk.Label(postgres_frame, text="PostgreSQL Conexion URL:").grid(row=0, column=0, sticky='w')
    postgres_entry = tk.Entry(postgres_frame)
    postgres_entry.grid(row=0, column=1)

    update_button = tk.Button(root, text="Actualizar Datos", command=actualizar_datos, font=("Helvetica", 16))
    update_button.place(relx=0.5, rely=0.75, anchor='n')

    last_update_time = tk.StringVar()
    tk.Label(root, text="Última actualización:", font=("Helvetica", 12)).place(relx=0.5, rely=0.85, anchor='n')
    tk.Label(root, textvariable=last_update_time, font=("Helvetica", 12)).place(relx=0.5, rely=0.9, anchor='n')

    # Iniciar la aplicación
    root.mainloop()
```

Conexión con Azure Blob Storage: La aplicación descarga archivos Parquet desde un contenedor de Azure Blob Storage utilizando Azure SDK y guarda los archivos con la siguiente definición.

```python
def download_blobs(account_url, container_name, sas_token, parquet_directory):
    container_client = ContainerClient(account_url=account_url, container_name=container_name, credential=sas_token)
    blobs_list = container_client.list_blobs()


    if not os.path.exists(parquet_directory):
        os.makedirs(parquet_directory)

    for blob in blobs_list:
        blob_client = container_client.get_blob_client(blob)
        download_path = os.path.join(parquet_directory, blob.name)
        if not os.path.exists(download_path):
            print(f"Descargando {blob.name}...")
            with open(download_path, "wb") as file:
                download_stream = blob_client.download_blob()
                file.write(download_stream.readall())
            print(f"Archivo {blob.name} descargado en {download_path}")


```

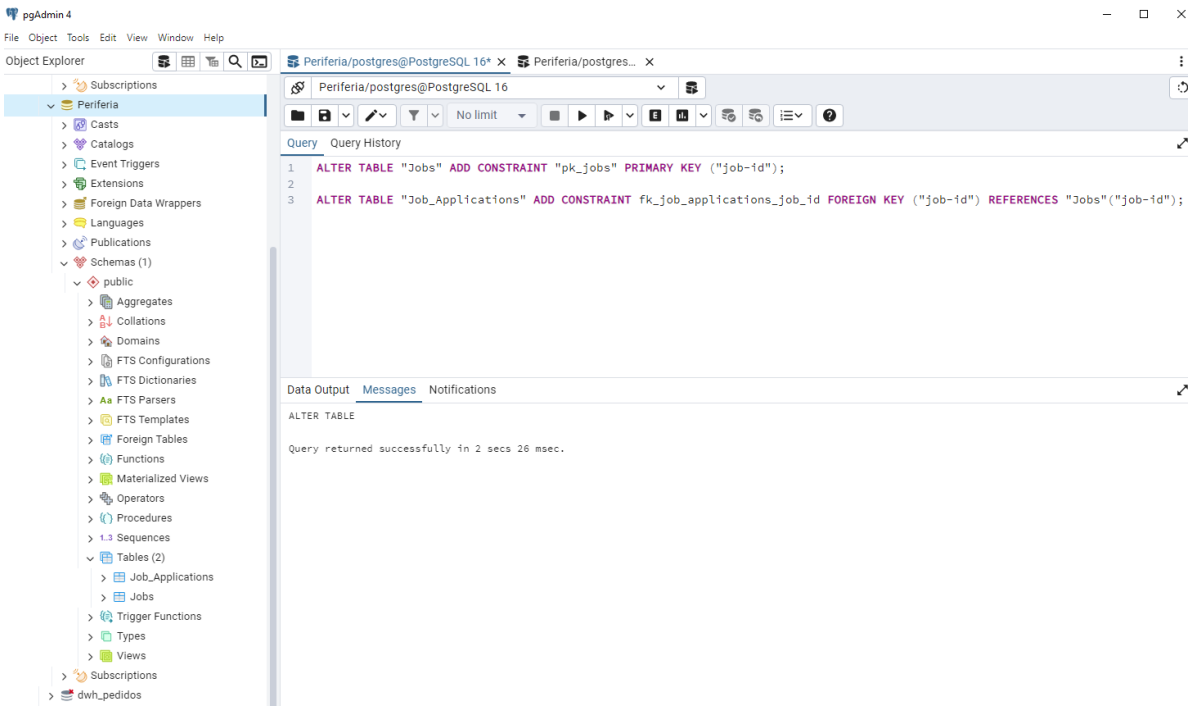
Conexión con PostgreSQL: Se conecta a una base de datos PostgreSQL utilizando psycopg2 y SQLAlchemy para almacenar los datos contenidos en los archivos Parquet descargados Manejo de Archivos: Los archivos se guardan fijados en la carpeta Downloads dentro del repositorio para construir la ruta relativa.

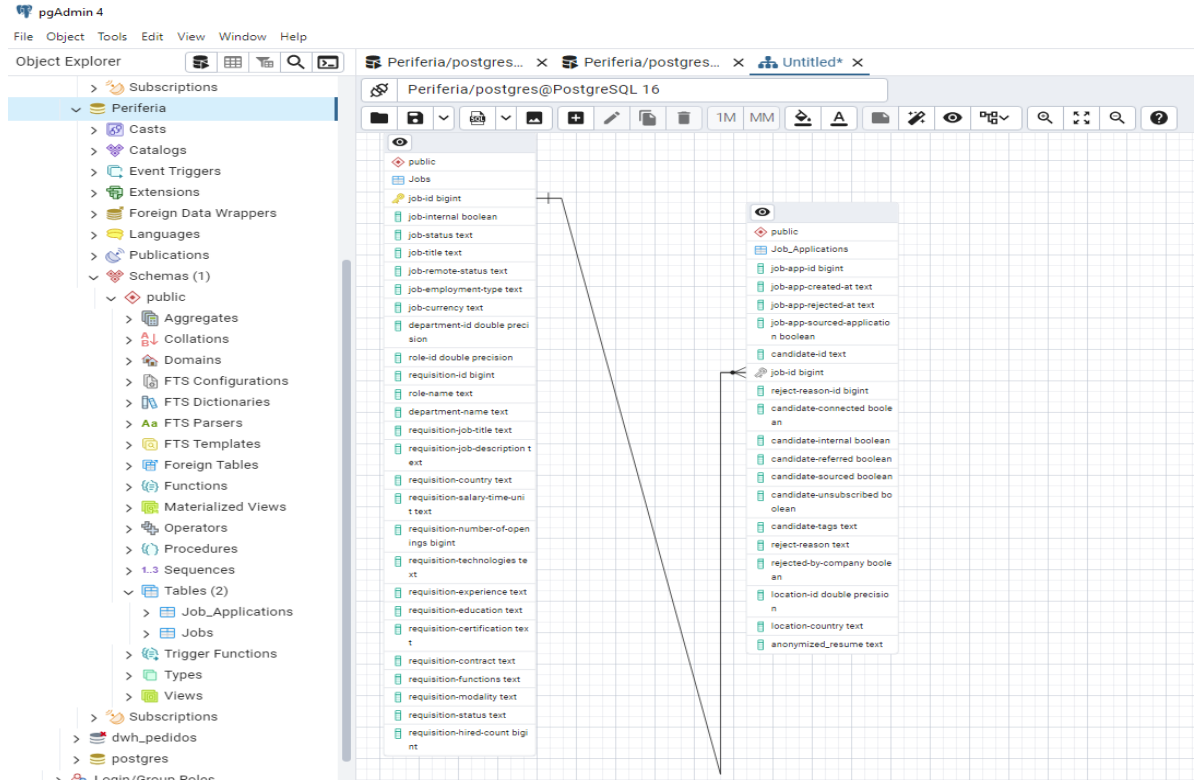
```python

    parquet_directory = os.path.join(os.path.dirname(__file__), 'Downloads')

```

Procesamiento de Archivos Parquet: Los archivos descargados se procesan con Pandas, y los datos se insertan o reemplazan en tablas dentro de PostgreSQL, se diseño también la actualización de estos mismo

```python
def process_parquet_files(parquet_directory, engine):
    if not os.path.exists(parquet_directory):
        raise FileNotFoundError(f"El directorio '{parquet_directory}' no existe.")

    for filename in os.listdir(parquet_directory):
        if filename.endswith('.parquet'):
            parquet_file = os.path.join(parquet_directory, filename)
            table_name = os.path.splitext(filename)[0]

            try:
                df = pd.read_parquet(parquet_file)
                print(f"Archivo Parquet '{filename}' leído exitosamente.")
                df.to_sql(table_name, engine, if_exists='replace', index=False)
                print(f"Datos importados exitosamente a la tabla '{table_name}'.")
            except Exception as e:
                print(f"Error al procesar el archivo '{filename}': {e}")


def actualizar_datos():
    azure_account_url = azure_account_entry.get()
    azure_container_name = azure_container_entry.get()
    azure_sas_token = azure_sas_token_entry.get()
    postgres_url = postgres_entry.get()

    engine = create_engine(postgres_url)

    download_blobs(azure_account_url, azure_container_name, azure_sas_token, parquet_directory)
    process_parquet_files(parquet_directory, engine)

    last_update_time.set(datetime.now().strftime('%Y-%m-%d %H:%M:%S'))
    messagebox.showinfo("Actualización Completa", "Los datos se han actualizado correctamente.")
```

Análisis PostgreSQL: Se verifica las tablas en la base de datos asignada “periferia” asegurándonos de que las tablas se importaron correctamente, se procede analizar los datos con que se cuenta y posterior mente procedemos a crear la relación de estas tablas una vez se identifiquen el campo en que estas se relacionan con sus respectivas llaves**.**





Librerías Utilizadas
A continuación, se describen las principales librerías utilizadas en este código:

- **Os:** Librería estándar de Python para interactuar con el sistema de archivos (crear directorios, verificar si un archivo existe, etc.).
- **Pandas:** Utilizada para leer los archivos Parquet y manipular los datos en formato DataFrame.
- **Psycopg2:** Permite la conexión directa y manipulación de una base de datos PostgreSQL. Se utiliza para crear una conexión a la base de datos.
- **SQLAlchemy:** Abstracción de alto nivel que permite realizar operaciones en la base de datos (en este caso, insertar DataFrames en PostgreSQL) sin tener que escribir SQL explícito.
- **Azure.storage.blob.ContainerClient:** Parte del SDK de Azure para acceder a contenedores en Azure Blob Storage. Permite listar, descargar y manipular archivos almacenados en la nube.
- **Tkinter:** Utilizado para crear la interfaz gráfica, permitiendo la interacción del usuario con la aplicación. Aquí se maneja la entrada de datos (credenciales de Azure y PostgreSQL), se muestra información y se activan las funciones de procesamiento de datos a través de botones.
- **PIL (Pillow):** Se utiliza para manejar y mostrar imágenes en la interfaz gráfica, en este caso, para la imagen de fondo.
- **Datetime:** Proporciona funciones para trabajar con fechas y horas. Aquí se utiliza para mostrar la última hora de actualización.

Manejo de Errores:
El código contiene manejo de errores básico, como la verificación de si un archivo ya ha sido descargado o si el directorio de destino existe. Esto ayuda a prevenir errores comunes, como la sobrescritura innecesaria de archivos o problemas de ruta.

Vista previa del script ejecutado:

```
PS C:\Users\Gilma\Documents\OperacionM30> & C:/Users/Gilma/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/Gilma/Documents/OperacionM30/azure_blob_to_postgres_updater.py
Archivo Parquet 'Jobs.parquet' leído exitosamente.
Datos importados exitosamente a la tabla 'Jobs'.
Archivo Parquet 'Job_Applications.parquet' leído exitosamente.
Datos importados exitosamente a la tabla 'Job_Applications'.
```