

PROTOCOLOS DE INTERNET

1 CUATRIMESTRE DE 2024

PRÁCTICA

Trabajo Práctico N.º 4 Parte A y B

Comisión: FN

Profesor/a: Javier Adolfo Ouret

Nº	Nombre y apellido	Carrera	DNI	Email
1	Christian Balderrama	Ing. Informática	42118900	christianbalderrama@uca.edu.ar
2	Fiorella Insfran Sanabria	Ing. Informática	96142187	fiorellainsfran@uca.edu.ar
3	Pablo Joaquin Cardozo	Ing. Informática	45178142a	cardozopabloj@uca.edu.ar
4	Pablo Joaquin Margewka	Ing. Informática	37754332	pablomargewka@uca.edu.ar
5	Fabián Leonardo De Simone	Ing. Informática	39433563	fdesimone96@uca.edu.ar

TP 4 - A :

Procedimiento

- Primero se realizó la instalación del programa Mosquitto MQTT
- Se utilizó el comando proporcionado por la cátedra para iniciar el programa “net start mosquitto” y para verificar que está en funcionamiento el comando “sc query mosquitto”.

```
NOMBRE_SERVICIO: mosquitto
        TIPO                : 10  WIN32_OWN_PROCESS
        ESTADO               : 4   RUNNING
                                (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
        CÓD_SALIDA_WIN32     : 0   (0x0)
        CÓD_SALIDA_SERVICIO : 0   (0x0)
        PUNTO_COMPROB.      : 0x0
        INDICACIÓN_INICIO   : 0x0

C:\Windows\System32>
```

- Luego se utilizó el siguiente comando en la cmd:
 - mosquitto_sub -h localhost -t sitio1/temperatura

Este comando sirve para suscribirse a un tema específico. A partir de este comando crearemos un suscriptor que estará escuchando y recibiendo datos provenientes del broker MQTT.

```
Instale la version mas reciente de PowerShell para obtener nuevas caracte
rísticas y mejoras. https://aka.ms/PSWindows

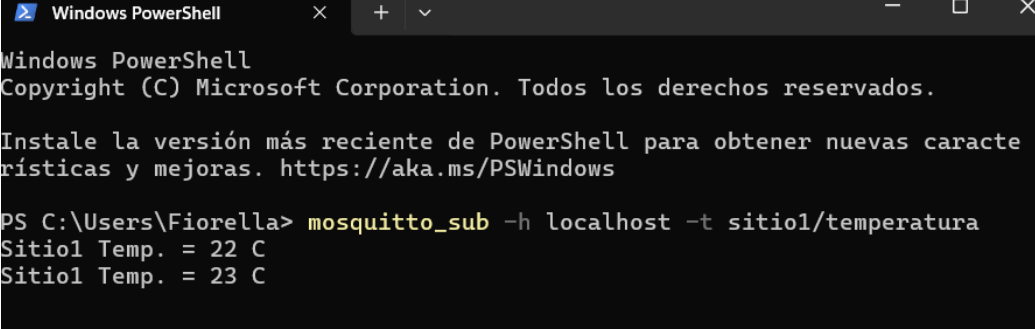
PS C:\Users\Fiorella> mosquitto_sub -h localhost -t sitio1/temperatura
```

- En otra terminal se corrieron los siguientes dos comandos:
 - mosquitto_pub -h localhost -t sitio1/temperatura -m "Sitio1 Temp. = 22 C".
 - mosquitto_pub -h localhost -t sitio1/temperatura -m "Sitio1 Temp. = 23 C".

Estos comandos corresponden a un publicador que envía estos registros a un broker MQTT para que este broker lo envíe a los suscriptores.

```
PS C:\Users\Fiorella> mosquitto_pub -h localhost -t sitio1/temperatura  
-m "Sitio1 Temp. = 22 C"  
PS C:\Users\Fiorella> mosquitto_pub -h localhost -t sitio1/temperatura  
-m "Sitio1 Temp. = 23 C"  
PS C:\Users\Fiorella> |
```

- Por último el resultado de la ejecución de los comandos que explicamos anteriormente son lo siguiente:



```
Windows PowerShell  
Copyright (C) Microsoft Corporation. Todos los derechos reservados.  
  
Instale la versión más reciente de PowerShell para obtener nuevas caracte  
rísticas y mejoras. https://aka.ms/PSWindows  
  
PS C:\Users\Fiorella> mosquitto_sub -h localhost -t sitio1/temperatura  
Sitio1 Temp. = 22 C  
Sitio1 Temp. = 23 C
```

En esta imagen se puede ver que el suscriptor recibe los datos provenientes del broker MQTT y los muestra por consola.

TP 4 - Parte B

Introducción

El objetivo de este trabajo práctico es crear un publicador MQTT que pueda leer los registros de una base de datos de sensores y publique estos registros, para que luego un suscriptor pueda captar estos registros y guardarlos en su propia base de datos.

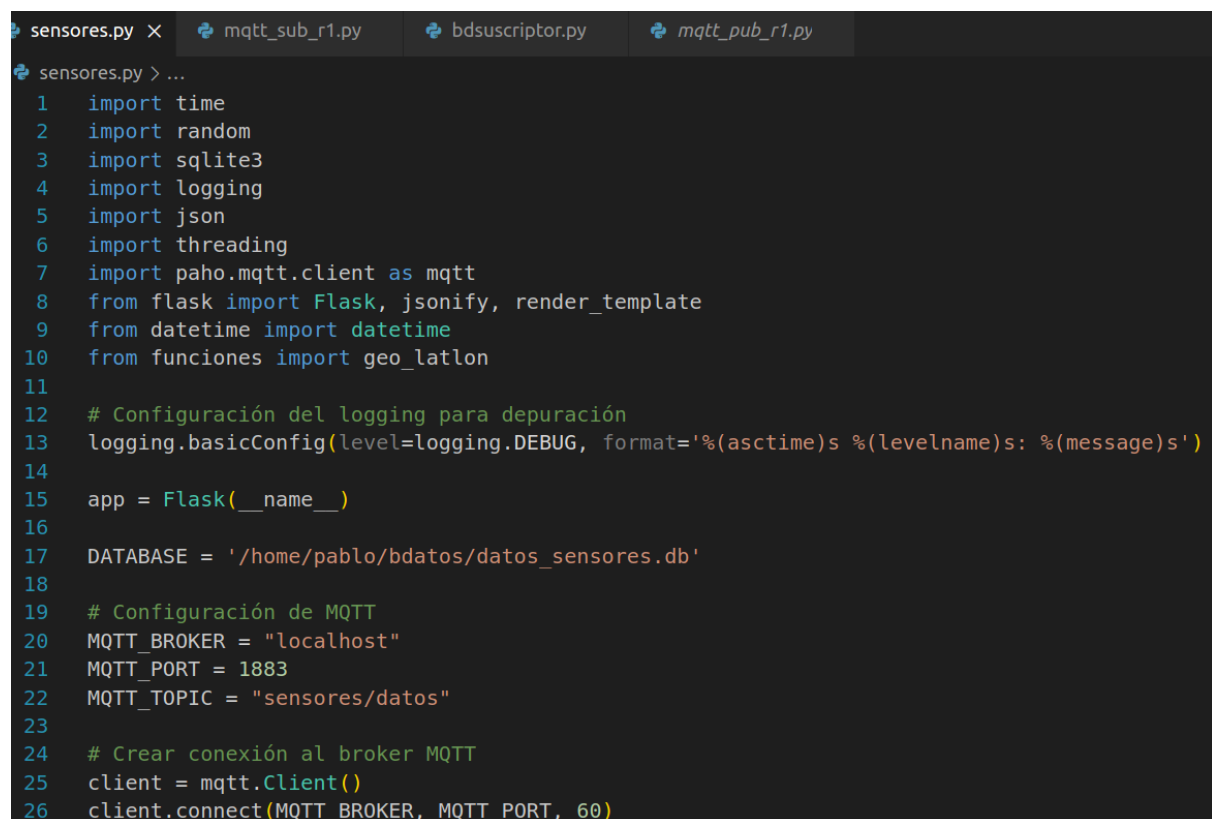
Procedimiento

En primer lugar lo que hicimos fue modificar lo realizado en el TP 3 y añadirle nuevas funcionalidades.

En el TP3 logramos insertar registros de los sensores y poder visualizarlos en formato de tabla desde <http://localhost:5000/>.

Modificamos este código y le agregamos una nueva función que es la del publicador MQTT. De esta manera, al ejecutar el programa, se iniciará el servidor Flask, y nos preguntará si queremos insertar un nuevo registro o si queremos iniciar el publicador MQTT. El publicador MQTT está configurado para leer cada 15 segundos la base de datos de sensores y publicar sus registros.

El código modificado es el siguiente:



```
sensores.py x  mqtt_sub_r1.py  bdsuscriptor.py  mqtt_pub_r1.py
sensores.py > ...
1  import time
2  import random
3  import sqlite3
4  import logging
5  import json
6  import threading
7  import paho.mqtt.client as mqtt
8  from flask import Flask, jsonify, render_template
9  from datetime import datetime
10 from funciones import geo_latlon
11
12 # Configuración del logging para depuración
13 logging.basicConfig(level=logging.DEBUG, format='%(asctime)s %(levelname)s: %(message)s')
14
15 app = Flask(__name__)
16
17 DATABASE = '/home/pablo/bdatos/datos_sensores.db'
18
19 # Configuración de MQTT
20 MQTT_BROKER = "localhost"
21 MQTT_PORT = 1883
22 MQTT_TOPIC = "sensores/datos"
23
24 # Crear conexión al broker MQTT
25 client = mqtt.Client()
26 client.connect(MQTT_BROKER, MQTT_PORT, 60)
```

```

sensores.py > ...
28 def create_table():
29     conn = sqlite3.connect(DATABASE)
30     cursor = conn.cursor()
31     cursor.execute('''CREATE TABLE IF NOT EXISTS lectura_sensores (
32         id INTEGER PRIMARY KEY,
33         co2 REAL,
34         temp REAL,
35         hum REAL,
36         fecha TEXT,
37         lugar TEXT,
38         altura REAL,
39         presion REAL,
40         presion_nm REAL,
41         temp_ext REAL
42     )''')
43     conn.commit()
44     conn.close()
45
46 @app.route('/')
47 def index():
48     return render_template('tabla_sensores_para_editar.html')
49

```

```

50 @app.route('/datos')
51 def datos():
52     conn = sqlite3.connect(DATABASE)
53     cursor = conn.cursor()
54     cursor.execute('SELECT * FROM lectura_sensores')
55     records = cursor.fetchall()
56     conn.close()
57     return jsonify([
58         {
59             'id': record[0],
60             'co2': record[1],
61             'temp': record[2],
62             'hum': record[3],
63             'fecha': record[4],
64             'lugar': record[5],
65             'altura': record[6],
66             'presion': record[7],
67             'presion_nm': record[8],
68             'temp_ext': record[9],
69         } for record in records])
70
71 def captura_datos():
72     temp_ext, presion, humedad_ext, descripcion_clima = geo_latlon()
73     print("Resultados= ", temp_ext, presion, humedad_ext, descripcion_clima)
74

```

```

74     while True:
75         try:
76             lugar = input("Lugar de la captura de los datos: ")
77             tipo_lugar = input("Tipo de lugar [au=abierto urbano] [an=abierto no urbano] [c=cerrado] ")
78             superficie = int(input("Superficie aproximada del lugar [m2]: "))
79             altura = int(input("Altura aproximada del lugar [m]: "))
80             presion_nm = presion
81             cant_capturas = int(input("Cantidad de capturas: "))
82             delta_t_capturas = int(input("Tiempo entre capturas (segs) : "))
83         except ValueError:
84             print("Error al ingresar datos...")
85             continue
86         else:
87             break
88

```

```

def captura_datos():
    cont = 0
    while cont < cant_capturas:
        cont += 1
        verdadero = 1
        if verdadero == 1:
            print("Datos Disponibles!")
            CO2_medido = random.uniform(250, 1100)
            temp_sensor = random.uniform(temp_ext, temp_ext + 10)
            humedad_relativa = random.uniform(40, 80)
            print("CO2: %d PPM" % CO2_medido)
            print("Temperatura: %0.2f degrees C" % temp_sensor)
            print("Humedad: %0.2f %% rH" % humedad_relativa)

            d = datetime.now()
            print("Fecha", d)
            timestampStr = d.strftime("%d-%b-%Y (%H:%M:%S.%f)")

            conn = sqlite3.connect(DATABASE)
            cursor = conn.cursor()
            cursor.execute('INSERT INTO lectura_sensores (co2, temp, hum, fecha, lugar, altura, presion,
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)',
            (CO2_medido, temp_sensor, humedad_relativa, timestampStr, lugar, altura, presion
            conn.commit()
            conn.close()

            print("Registro insertado..., acumulados:", cont, "\n")
            time.sleep(delta_t_capturas)
            print("\nEsperando nuevo registro de datos ...\n")

    print("Cierro conexión ...")

```

```

120 def publish_data():
121     while True:
122         try:
123             conn = sqlite3.connect(DATABASE)
124             (variable) records: list[Any] lectura_sensores')
125             records = cursor.fetchall()
126             conn.close()
127
128             for record in records:
129                 data = {
130                     'id': record[0],
131                     'co2': record[1],
132                     'temp': record[2],
133                     'hum': record[3],
134                     'fecha': record[4],
135                     'lugar': record[5],
136                     'altura': record[6],
137                     'presion': record[7],
138                     'presion_nm': record[8],
139                     'temp_ext': record[9]
140                 }
141                 client.publish(MQTT_TOPIC, json.dumps(data))
142                 logging.debug(f"Datos publicados: {data}")
143
144             # Espera 15 segundos antes de la siguiente publicación
145             time.sleep(15)
146         except Exception as e:
147             logging.error(f"Error al publicar datos: {e}")
148             time.sleep(5)
149

```

```

0
1 if __name__ == '__main__':
2     create_table()
3
4     # Iniciar el servidor Flask
5     logging.info("Iniciando servidor Flask...")
6     flask_thread = threading.Thread(target=app.run, kwargs={'host': '0.0.0.0', 'port': 5000})
7     flask_thread.start()
8
9     # Mostrar el menú principal
10    while True:
11        if flask_thread.is_alive():
12            break
13        time.sleep(1)
14
15    while True:
16        print("\nMENU:")
17        print("1. Iniciar MQTT (Publicador)")
18        print("2. Agregar Registro Manualmente")
19        print("3. Salir")
20
21        opcion = input("Seleccione una opción (1/2/3): ")
22
23        if opcion == "1":
24            logging.info("Iniciando publicador MQTT...")
25            thread_publish = threading.Thread(target=publish_data)
26            thread_publish.start()
27

```

```

177
178         elif opcion == "2":
179             logging.info("Iniciando captura de datos...")
180             captura_datos()
181
182         elif opcion == "3":
183             break
184
185         else:
186             print("Opción no válida. Intente de nuevo.")
187

```

Posteriormente, como necesitamos una nueva base de datos para que el suscriptor guarde lo que recibe del publicador, creamos un pequeño programa en python que se encarga de crear una nueva base de datos con las tablas y campos necesarios.

```

bdsuscriptor.py > ...
1  import sqlite3
2
3  NEW_DATABASE = '/home/pablo/Downloads/TP4/bdsuscriptor/suscriptor.db'
4
5  def create_suscriptor_table():
6      conn = sqlite3.connect(NEW_DATABASE)
7      cursor = conn.cursor()
8      cursor.execute('''CREATE TABLE IF NOT EXISTS lectura_sensores (
9                          id INTEGER PRIMARY KEY,
10                         co2 REAL,
11                         temp REAL,
12                         hum REAL,
13                         fecha TEXT,
14                         lugar TEXT,
15                         altura REAL,
16                         presion REAL,
17                         presion_nm REAL,
18                         temp_ext REAL
19                     )''')
20      conn.commit()
21      conn.close()
22
23  # Llamar a la función para crear la tabla si no existe
24  create_suscriptor_table()
25

```

Posteriormente, utilizamos el archivo base llamado mqtt_sub_r1.py, indicando el nuevo directorio en donde se encuentra nuestra nueva base de datos. Por otra parte, como el publicador publica cada 15 segundos y publica todos los registros de la base de datos, necesitamos una forma de evitar que haya registros duplicados. Por este motivo, añadimos un condicional que verifica las fechas para evitar que esto suceda.

El código del suscriptor es el siguiente


```

mqtt_sub_r1.py > ...
1  import logging
2  import sqlite3
3  import paho.mqtt.client as mqtt
4  import json
5  from datetime import datetime, timedelta
6
7  logging.basicConfig(level=logging.DEBUG, format='%(asctime)s %(levelname)s: %(message)s')
8
9  MQTT_BROKER = "localhost"
10 MQTT_PORT = 1883
11 MQTT_TOPIC = "sensores/datos"
12
13 NEW_DATABASE = '/home/pablo/Downloads/TP4/bdsuscriptor/suscriptor.db'
14
15 def on_connect(client, userdata, flags, rc):
16     if rc == 0:
17         logging.info("Conectado al broker MQTT")
18         client.subscribe(MQTT_TOPIC)
19     else:
20         logging.error(f"Conexión fallida con código de resultado: {rc}")
21
22 def on_message(client, userdata, msg):
23     try:
24         data = json.loads(msg.payload.decode())
25         if not is_duplicate_date(data['fecha']): # Verificar si existe
26             insert_into_database(data)
27     except Exception as e:
28         logging.error(f"Error al procesar mensaje: {str(msg.payload)} - {e}")
29

```

```

29
30 def is_duplicate_date(new_date):
31     try:
32         conn = sqlite3.connect(NEW_DATABASE)
33         cursor = conn.cursor()
34
35         cursor.execute("SELECT COUNT(*) FROM lectura_sensores WHERE fecha = ?", (new_date,))
36         count = cursor.fetchone()[0]
37
38         conn.close()
39
40         # Si count > 0, significa que ya existe un registro con esa fecha
41         return count > 0
42     except Exception as e:
43         logging.error(f"Error al verificar duplicados por fecha: {e}")
44         return False
45
46 def insert_into_database(data):
47     try:
48         conn = sqlite3.connect(NEW_DATABASE)
49         cursor = conn.cursor()
50         cursor.execute('''INSERT INTO lectura_sensores (co2, temp, hum, fecha, lugar, altura, presion, pres
51             VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)''',
52             (data['co2'], data['temp'], data['hum'], data['fecha'], data['lugar'], data['altura'],
53             data['presion'], data['presion_nm'], data['temp_ext']))
54         conn.commit()
55         conn.close()
56         logging.debug("Datos insertados en la base de datos del suscriptor")
57     except Exception as e:
58         logging.error(f"Error al insertar datos en la base de datos del suscriptor: {e}")
59

```

```

59
60 client = mqtt.Client()
61 client.on_connect = on_connect
62 client.on_message = on_message
63
64 try:
65     client.connect(MQTT_BROKER, MQTT_PORT, 60)
66     logging.info("Conectando al broker MQTT")
67     client.loop_forever()
68 except Exception as e:
69     logging.error(f"Error al conectar al broker MQTT: {e}")
70

```

```

❏ mqtt_sub_r1.py > ...
1  import logging
2  import sqlite3
3  import paho.mqtt.client as mqtt
4  import json
5  from datetime import datetime, timedelta
6
7  logging.basicConfig(level=logging.DEBUG, format='%(asctime)s %(levelname)s: %(message)s')
8
9  MQTT_BROKER = "localhost"
10 MQTT_PORT = 1883
11 MQTT_TOPIC = "sensores/datos"
12
13 NEW_DATABASE = '/home/pablo/Downloads/TP4/bdsuscriptor/suscriptor.db'
14
15 def on_connect(client, userdata, flags, rc):
16     if rc == 0:
17         logging.info("Conectado al broker MQTT")
18         client.subscribe(MQTT_TOPIC)
19     else:
20         logging.error(f"Conexión fallida con código de resultado: {rc}")
21
22 def on_message(client, userdata, msg):
23     try:
24         data = json.loads(msg.payload.decode())
25         if not is_duplicate_date(data['fecha']): # Verificar si la fecha ya ha sido procesada
26             insert_into_database(data)
27     except Exception as e:
28         logging.error(f"Error al procesar mensaje: {str(msg.payload)} - {e}")
29

```

```

30 def is_duplicate_date(new_date):
31     try:
32         conn = sqlite3.connect(NEW_DATABASE)
33         cursor = conn.cursor()
34
35         # Aquí puedes ajustar la consulta para buscar fechas similares o idénticas
36         cursor.execute("SELECT COUNT(*) FROM lectura_sensores WHERE fecha = ?", (new_date,))
37         count = cursor.fetchone()[0]
38
39         conn.close()
40
41         # Si count > 0, significa que ya existe un registro con esa fecha
42         return count > 0
43     except Exception as e:
44         logging.error(f"Error al verificar duplicados por fecha: {e}")
45     return False
46
47 def insert_into_database(data):
48     try:
49         conn = sqlite3.connect(NEW_DATABASE)
50         cursor = conn.cursor()
51         cursor.execute('INSERT INTO lectura_sensores (co2, temp, hum, fecha, lugar, altura, presion, presion_nm, temp_ext) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)',
52                        (data['co2'], data['temp'], data['hum'], data['fecha'], data['lugar'], data['altura'],
53                        data['presion'], data['presion_nm'], data['temp_ext']))
54
55         conn.commit()
56         conn.close()
57         logging.debug("Datos insertados en la base de datos del suscriptor")
58     except Exception as e:
59         logging.error(f"Error al insertar datos en la base de datos del suscriptor: {e}")

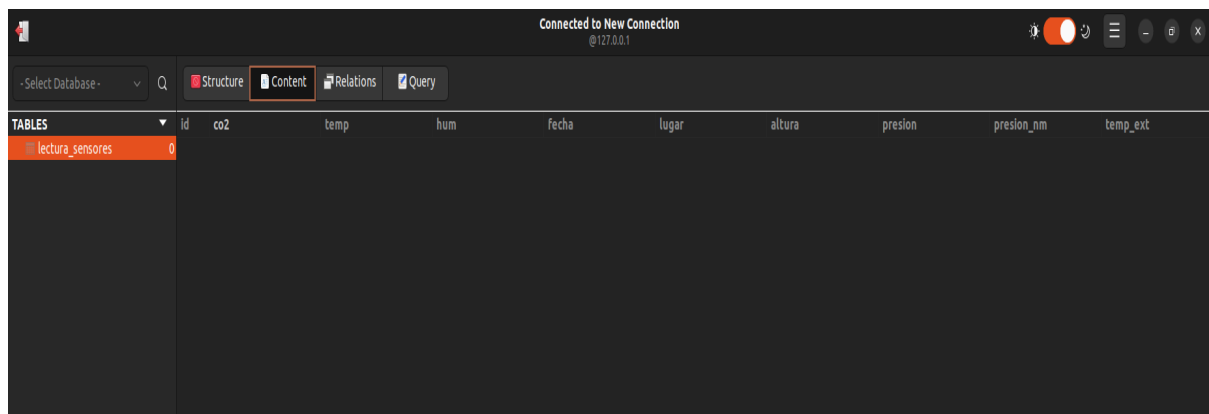
```

Funcionamiento

- 1) En primer lugar iniciamos el broker MQTT con `sudo systemctl start mosquitto`

```
pablo@pablo-OMEN-by-HP-Laptop-15-ce0xx: ~/Downloads/TP4
pablo@pablo-OMEN-by-HP-Laptop-15-ce0xx:~/Downloads/TP4$ sudo systemctl start mosquitto
[sudo] password for pablo:
pablo@pablo-OMEN-by-HP-Laptop-15-ce0xx:~/Downloads/TP4$
```

- 2) Utilizamos Sequeler para gestionar la base de datos creada para el suscriptor. Esto nos permitirá consultar rápidamente los registros de esta tabla para ver si se agregaron nuevos registros y además poder realizar consultas SQL. Para probar el funcionamiento, inicialmente dejamos la tabla sin registros



- 3) Iniciamos el suscriptor. Podemos observar que se conecta al broker MQTT y queda esperando para recibir nuevos datos.

```
pablo@pablo-OMEN-by-HP-Laptop-15-ce0xx:~/Downloads/TP4$ python3 mqtt_sub_r1.py
/home/pablo/Downloads/TP4/mqtt_sub_r1.py:60: DeprecationWarning: Callback API version 1 is deprecated, update to latest version
  client = mqtt.Client()
2024-06-29 17:37:19,666 INFO: Conectando al broker MQTT
2024-06-29 17:37:19,666 INFO: Conectado al broker MQTT
```

- 4) Iniciamos el publicador. Inicialmente encontraremos un menú para poder seleccionar si queremos agregar un nuevo registro a la base de datos de sensores o si queremos iniciar el publicador MQTT

```
MENU:
1. Iniciar MQTT (Publicador)
2. Agregar Registro Manualmente
3. Salir
Seleccione una opción (1/2/3):
```

Una vez presionado 1, se iniciará el publicador MQTT

```
MENU:
1. Iniciar MQTT (Publicador)
2. Agregar Registro Manualmente
3. Salir
Seleccione una opción (1/2/3): 1
2024-06-29 17:42:30,740 INFO: Iniciando publicador MQTT...
2024-06-29 17:42:30,741 DEBUG: Datos publicados: {'id': 1, 'co2': 486.20950998336764, 'temp': 23.25409132772101, 'hum': 68.76470663188968, 'fecha': '10-Jun-2024 (16:29:32.318864)', 'lugar': 'Nagoya', 'altura': 20.0, 'presion': 1008.0, 'presion_nm': 1008.0, 'temp_ext': 20.8}
2024-06-29 17:42:30,742 DEBUG: Datos publicados: {'id': 2, 'co2': 574.8414758697535, 'temp': 24.513123129363454, 'hum': 58.91699470804806, 'fecha': '10-Jun-2024 (16:29:42.489696)', 'lugar': 'Nagoya', 'altura': 20.0, 'presion': 1008.0, 'presion_nm': 1008.0, 'temp_ext': 20.8}
2024-06-29 17:42:30,742 DEBUG: Datos publicados: {'id': 3, 'co2': 951.8638225660931, 'temp': 24.012149220255743, 'hum': 73.93181830353811, 'fecha': '10-Jun-2024 (16:29:52.562541)', 'lugar': 'Nagoya', 'altura': 20.0, 'presion': 1008.0, 'presion_nm': 1008.0, 'temp_ext': 20.8}
2024-06-29 17:42:30,742 DEBUG: Datos publicados: {'id': 4, 'co2': 926.5356155627225, 'temp': 27.32106005781399, 'hum': 48.62165385288229, 'fecha': '10-Jun-2024 (16:30:02.707592)', 'lugar': 'Nagoya', 'altura': 20.0, 'presion': 1008.0, 'presion_nm': 1008.0, 'temp_ext': 20.8}
2024-06-29 17:42:30,742 DEBUG: Datos publicados: {'id': 5, 'co2': 488.49868502759114, 'temp': 28.691433741226955, 'hum': 76.62137108776943, 'fecha': '10-Jun-2024 (16:30:12.756432)', 'lugar': 'Nagoya', 'altura': 20.0, 'presion': 1008.0, 'presion_nm': 1008.0, 'temp_ext': 20.8}
2024-06-29 17:42:30,742 DEBUG: Datos publicados: {'id': 6, 'co2': 751.4081056717018, 'temp': 26.874715174972778, 'hum': 47.97023889050861, 'fecha': '10-Jun-2024 (16:30:22.814985)', 'lugar': 'Nagoya', 'altura': 20.0, 'presion': 1008.0, 'presion_nm': 1008.0, 'temp_ext': 20.8}
2024-06-29 17:42:30,742 DEBUG: Datos publicados: {'id': 7, 'co2': 564.2137205467362, 'temp': 26.40900327484611, 'hum': 57.228350554055844, 'fecha': '10-Jun-2024 (16:30:32.865128)', 'lugar': 'Nagoya', 'altura': 20.0, 'presion': 1008.0, 'presion_nm': 1008.0, 'temp_ext': 20.8}
2024-06-29 17:42:30,742 DEBUG: Datos publicados: {'id': 8, 'co2': 448.0943464852103, 'temp': 24.63903521664107, 'hum': 58.319697126266234, 'fecha': '10-Jun-2024 (16:30:42.932154)', 'lugar': 'Nagoya', 'altura': 20.0, 'presion': 1008.0, 'presion_nm': 1008.0, 'temp_ext': 20.8}
2024-06-29 17:42:30,742 DEBUG: Datos publicados: {'id': 9, 'co2': 890.3078540396792, 'temp': 21.296969365342097, 'hum': 43.82479030813009, 'fecha': '10-Jun-2024 (16:30:52.998533)', 'lugar': 'Nagoya', 'altura': 20.0, 'presion': 1008.0, 'presion_nm': 1008.0, 'temp_ext': 20.8}
2024-06-29 17:42:30,742 DEBUG: Datos publicados: {'id': 10, 'co2': 1000.6983276150985, 'temp': 30.08481687786497, 'hum': 51.27394109822281, 'fecha': '10-Jun-2024 (16:31:03.054603)', 'lugar': 'Nagoya', 'altura': 20.0, 'presion': 1008.0, 'presion_nm': 1008.0, 'temp_ext': 20.8}
2024-06-29 17:42:30,742 DEBUG: Datos publicados: {'id': 11, 'co2': 1081.4431187055873, 'temp': 46.18928951124835, 'hum': 72.94761284676017, 'fecha': '29-Jun-2024 (16:20:28.276896)', 'lugar': 'Korea', 'altura': 1996691.0, 'presion': 1011.0, 'presion_nm': 1011.0, 'temp_ext': 36.23}
2024-06-29 17:42:30,742 DEBUG: Datos publicados: {'id': 12, 'co2': 828.7648636100574, 'temp': 40.50091180578116, 'hum': 41.48285980894017, 'fecha': '29-Jun-2024 (16:20:31.344791)', 'lugar': 'Korea', 'altura': 1996691.0, 'presion': 1011.0, 'presion_nm': 1011.0, 'temp_ext': 36.23}
2024-06-29 17:42:30,743 DEBUG: Datos publicados: {'id': 13, 'co2': 671.7192168607246, 'temp': 36.78735875060728, 'hum': 74.99453032680952, 'fecha': '29-Jun-2024 (16:50:37.783851)', 'lugar': 'Paris', 'altura': 29484.0, 'presion': 1010.0, 'presion_nm': 1010.0, 'temp_ext': 30.99}
2024-06-29 17:42:30,743 DEBUG: Datos publicados: {'id': 14, 'co2': 709.5138467651213, 'temp': 38.72943875694868, 'hum': 48.16805823918049, 'fecha': '29-Jun-2024 (16:50:40.841751)', 'lugar': 'Paris', 'altura': 29484.0, 'presion': 1010.0, 'presion_nm': 1010.0, 'temp_ext': 30.99}
```

El publicador MQTT leyó los registros de la base de datos de sensores y los publicó, mostrándolos por consola.

Utilizando el DBeaver CE, podemos observar que los datos mostrados por el publicador son los mismos que estaban publicados en la base de datos de sensores

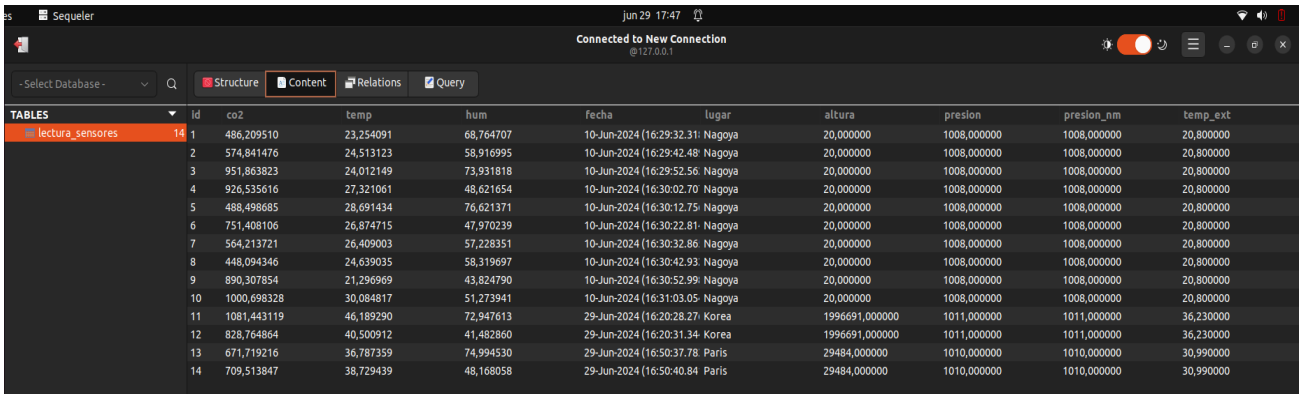
DBeaver 24.0.5 - lectura_sensores											
File Edit Navigate Search SQL Editor Database Window Help											
SQL Commit Rollback Auto datos_sensores.db <N/A>											
lectura_sensores x id datos_sensores.db co2 LECTURA_SENSORES_PK id presion lugar altura											
Properties Data ER Diagram											
lectura_sensores Enter a SQL expression to filter results (use Ctrl+Space)											
	id	co2	temp	hum	fecha	lugar	altura	presion	presion_nm	temp_ext	
1	1	486.2095099834	23.2540913277	68.7647066319	10-Jun-2024 (16:29:32.318864)	Nagoya	20	1,008	1,008	20.8	
2	2	574.8414758698	24.5131231294	58.9169947086	10-Jun-2024 (16:29:42.489696)	Nagoya	20	1,008	1,008	20.8	
3	3	951.8638225661	24.0121492203	73.9318183035	10-Jun-2024 (16:29:52.562541)	Nagoya	20	1,008	1,008	20.8	
4	4	926.5356155627	27.3210608578	48.6216538529	10-Jun-2024 (16:30:02.707592)	Nagoya	20	1,008	1,008	20.8	
5	5	488.4986850276	28.6914337412	76.6213710878	10-Jun-2024 (16:30:12.756432)	Nagoya	20	1,008	1,008	20.8	
6	6	751.4081056717	26.874715175	47.9702388905	10-Jun-2024 (16:30:22.814985)	Nagoya	20	1,008	1,008	20.8	
7	7	564.2137205467	26.4090032748	57.2283505541	10-Jun-2024 (16:30:32.865128)	Nagoya	20	1,008	1,008	20.8	
8	8	448.0943464852	24.6390352166	58.3196971263	10-Jun-2024 (16:30:42.932154)	Nagoya	20	1,008	1,008	20.8	
9	9	890.3078540397	21.2969693653	43.8247903081	10-Jun-2024 (16:30:52.998533)	Nagoya	20	1,008	1,008	20.8	
10	10	1,000.6983276151	30.0848168779	51.2739410982	10-Jun-2024 (16:31:03.054603)	Nagoya	20	1,008	1,008	20.8	
11	11	1,081.4431187056	46.1892895112	72.9476128468	29-Jun-2024 (16:20:28.276896)	Korea	1,996.691	1,011	1,011	36.23	
12	12	828.7648636101	40.5009118058	41.4828598089	29-Jun-2024 (16:20:31.344791)	Korea	1,996.691	1,011	1,011	36.23	
13	13	671.7192160607	36.7873587551	74.9945303268	29-Jun-2024 (16:50:37.783851)	Paris	29,484	1,010	1,010	30.99	
14	14	709.5138467651	38.7294387569	48.1680582392	29-Jun-2024 (16:50:40.841751)	Paris	29,484	1,010	1,010	30.99	

El publicador va a leer y mostrar todos los registros que encontró en la base de datos de sensores. Realiza lo mismo cada 15 segundos.

- 5) El suscriptor que estaba escuchando, recibe los datos provenientes del publicador y los envía a su propia base de datos.

```
pablo@pablo-OMEN-by-HP-Laptop-15-ce0xx: ~/Downloads/TP4
pablo@pablo-OMEN-by-HP-Laptop-15-ce0xx:~/Downloads/TP4$ python3 mqtt_sub_r1.py
/home/pablo/Downloads/TP4/mqtt_sub_r1.py:60: DeprecationWarning: Callback API version 1 is deprecated, update to latest version
client = mqtt.Client()
2024-06-29 17:45:33,868 INFO: Conectando al broker MQTT
2024-06-29 17:45:33,869 INFO: Conectado al broker MQTT
2024-06-29 17:45:38,047 DEBUG: Datos insertados en la base de datos del suscriptor
2024-06-29 17:45:38,105 DEBUG: Datos insertados en la base de datos del suscriptor
2024-06-29 17:45:38,579 DEBUG: Datos insertados en la base de datos del suscriptor
2024-06-29 17:45:38,880 DEBUG: Datos insertados en la base de datos del suscriptor
2024-06-29 17:45:38,930 DEBUG: Datos insertados en la base de datos del suscriptor
2024-06-29 17:45:38,980 DEBUG: Datos insertados en la base de datos del suscriptor
2024-06-29 17:45:39,030 DEBUG: Datos insertados en la base de datos del suscriptor
2024-06-29 17:45:39,080 DEBUG: Datos insertados en la base de datos del suscriptor
2024-06-29 17:45:39,139 DEBUG: Datos insertados en la base de datos del suscriptor
2024-06-29 17:45:39,197 DEBUG: Datos insertados en la base de datos del suscriptor
2024-06-29 17:45:39,255 DEBUG: Datos insertados en la base de datos del suscriptor
2024-06-29 17:45:39,314 DEBUG: Datos insertados en la base de datos del suscriptor
2024-06-29 17:45:39,372 DEBUG: Datos insertados en la base de datos del suscriptor
2024-06-29 17:45:39,430 DEBUG: Datos insertados en la base de datos del suscriptor
```

6) Posteriormente revisamos la base de datos del suscriptor con Sequeler y podemos confirmar que nuestro suscriptor recibe correctamente todos los registros que le envió el broker MQTT y los envió a su propia base de datos.



The screenshot shows the Sequencer application interface. The 'Content' tab is selected, displaying a table named 'lectura_sensores'. The table has 14 rows of data, each representing a sensor reading. The columns are: id, co2, temp, hum, fecha, lugar, altura, presion, presion_nm, and temp_ext.

id	co2	temp	hum	fecha	lugar	altura	presion	presion_nm	temp_ext
1	486,209510	23,254091	68,764707	10-Jun-2024 (16:29:32.31)	Nagoya	20,000000	1008,000000	1008,000000	20,800000
2	574,841476	24,513123	58,916995	10-Jun-2024 (16:29:42.48)	Nagoya	20,000000	1008,000000	1008,000000	20,800000
3	951,863823	24,012149	73,931818	10-Jun-2024 (16:29:52.56)	Nagoya	20,000000	1008,000000	1008,000000	20,800000
4	926,535616	27,321061	48,621654	10-Jun-2024 (16:30:02.70)	Nagoya	20,000000	1008,000000	1008,000000	20,800000
5	488,498685	28,691434	76,621371	10-Jun-2024 (16:30:12.75)	Nagoya	20,000000	1008,000000	1008,000000	20,800000
6	751,408106	26,874715	47,970239	10-Jun-2024 (16:30:22.81)	Nagoya	20,000000	1008,000000	1008,000000	20,800000
7	564,213721	26,409003	57,228351	10-Jun-2024 (16:30:32.86)	Nagoya	20,000000	1008,000000	1008,000000	20,800000
8	448,094346	24,639035	58,319697	10-Jun-2024 (16:30:42.93)	Nagoya	20,000000	1008,000000	1008,000000	20,800000
9	890,307854	21,296969	43,824790	10-Jun-2024 (16:30:52.99)	Nagoya	20,000000	1008,000000	1008,000000	20,800000
10	1000,698328	30,084817	51,273941	10-Jun-2024 (16:31:03.05)	Nagoya	20,000000	1008,000000	1008,000000	20,800000
11	1081,443119	46,189290	72,947613	29-Jun-2024 (16:20:28.27)	Korea	1996691,000000	1011,000000	1011,000000	36,230000
12	828,764864	40,500912	41,482860	29-Jun-2024 (16:20:31.34)	Korea	1996691,000000	1011,000000	1011,000000	36,230000
13	671,719216	36,787359	74,994530	29-Jun-2024 (16:50:37.78)	Paris	29484,000000	1010,000000	1010,000000	30,990000
14	709,513847	38,729439	48,168058	29-Jun-2024 (16:50:40.84)	Paris	29484,000000	1010,000000	1010,000000	30,990000

Conclusión

A partir de este trabajo práctico, pudimos observar el funcionamiento del sistema MQTT. En la parte A, partimos de un funcionamiento básico de unos pocos comandos en la terminal para iniciar el servicio, publicar un dato y ver cómo el suscriptor escucha, lee y muestra los datos recibidos. En la parte B, aumentamos la complejidad para asimilar a un caso de uso real, en donde el publicador MQTT lee desde una base de datos y envía todos los registros a los clientes que se suscriban. Se utilizó como base el TP3 realizado anteriormente, que nos permitió conectarnos a una base de datos y guardar los registros.