Università
degli Studi di
Messina

Database project

Report

Course: Database module NoSQL

Professor: Armando Ruggeri

Student: Tairbek Akhayev, 551094

Introduction

This report describes the process of developing and implementing a Python script to collect, process and distribute IoT data across three different databases (MongoDB, Neo4j and MySQL) using the Mosquitto MQTT server. The main goal is to create a system for storing IoT device data such as device ID, timestamp, motion state, coordinates and connection parameters.

Technologies and tools used

Mosquitto MQTT server: A protocol for transferring messages between devices and the server.

Python: A programming language for implementing data processing logic.

Paho MQTT: A library for working with MQTT in Python.

MongoDB: A document-oriented database for storing device data.

Neo4j: A graph database for storing relationships between devices and data types.

MySQL: Relational database for storing structured data.

Data Description

The data is a collection of JSON objects, each of which contains information about different IoT devices. Each object includes several keys that provide detailed information about the device's state, location, and connection type. Below is a detailed description of the data structure of each JSON object.


```
{"device_id": "sensor_004", "timestamp": "2024-06-17T11:15:19.203155", "motion": false, "location": {"latitude": 32.604991, "longitude": -132.992578}, "data": {"data_type": {"type": "Bluetooth", "ssid": "Bluetooth_Device1", "signal_strength": -14}}}
```

Data structure

device_id: Unique device identifier.

Example: "device_id": "sensor_004"

timestamp: A timestamp indicating the time the data was received.

Example: "timestamp": "2024-06-17T11:15:19.203155"

motion: A Boolean value indicating the presence of motion.

Example: "motion": false

location: An object containing information about the geographic location of the device.

latitude: Latitude of the location.

Example: "latitude": 32.604991

longitude: Longitude of the location.

Example: "longitude": -132.992578

data: An object containing information about the connection type and related parameters.

data_type: An object containing information about the connection type.

type: Connection type (for example, Bluetooth, WiFi, Zigbee).

Example: "type": "Bluetooth"

ssid: Network or device identifier.

Example: "ssid": "Bluetooth_Device1"

signal_strength: Signal strength.

Example: "signal_strength": -14

Data generation

This data was generated using the following Python script. The script creates a list of dictionaries, each of which represents a record of data about an IoT device, and saves them in JSON format.

```python
import json
import random
from datetime import datetime, timedelta

# Список типов и их соответствующих сетей
network_types = [
    {"type": "WiFi", "ssid": ["WiFi_Network", "Home_Network", "Office_Network"]},
    {"type": "Bluetooth", "ssid": ["Bluetooth_Device1", "Bluetooth_Device2", "Bluetooth_Device3"]},
    {"type": "Zigbee", "ssid": ["Zigbee_Device1", "Zigbee_Device2", "Zigbee_Device3"]}
]

# Функция для генерации случайных данных
def generate_data(num_records=100):
    devices = [f"sensor_{str(i).zfill(3)}" for i in range(1, 101)]

    data = []
    start_time = datetime.now()

    for _ in range(num_records):
        device_id = random.choice(devices)
        timestamp = start_time + timedelta(seconds=random.randint(0, 3600))
        motion = random.choice([True, False])
        location = {
            "latitude": round(random.uniform(-90.0, 90.0), 6),
            "longitude": round(random.uniform(-180.0, 180.0), 6)
        }
        network_type = random.choice(network_types)
        ssid = random.choice(network_type["ssid"])

        data_entry = {
            "device_id": device_id,
            "timestamp": timestamp.isoformat(),
            "motion": motion,
            "location": location,
            "data": {
                "data_type": {
                    "type": network_type["type"],
                    "ssid": ssid,
                    "signal_strength": random.randint(-100, 0)
                }
            },
        }

        data.append(data_entry)

    return data

# Сохранение данных в файл
def save_data_to_file(data, filename="iot_data.json"):
    with open(filename, "w") as file:
        for record in data:
            file.write(json.dumps(record) + "\n")

# Генерация и сохранение данных
data = generate_data(num_records=100)
save_data_to_file(data)
print("Data generated and saved to iot_data.json")
```

Project structure

1. Connecting to databases

1.1 MongoDB

```
mongo_client = MongoClient('mongodb://localhost:27017/')
db = mongo_client['iotdb']
collection = db['iot']
```

1.2 Neo4j

```
neo4j_graph = Graph("bolt://localhost:7687", auth=("neo4j", "password"))
```

1.3 MySQL

```
def connect_to_mysql():
    mysql_conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password="",
        database="iot"
    )
    return mysql_conn
```

2. Setting up an MQTT client

2.1 Connecting to MQTT server

```
mqtt_host = "localhost"
mqtt_port = 1883
mqtt_topic = "sensors/devices"
```

```
mqtt_client.connect("localhost", 1883, 60)
```

The publish_data function is designed to publish data from a file to an MQTT server topic using a specified packet size. Below is a detailed description of each step of the function.

```python
def publish_data(filename, batch_size=10):
    try:
        # Открытие файла и чтение данных
        with open(filename, 'r', encoding='utf-8') as file:
            data = [json.loads(line) for line in file if line.strip()]  # Чтение каждой строки как JSON-объек

        # Разделение данных на пакеты
        for i in range(0, len(data), batch_size):
            batch = data[i:i + batch_size]
            mqtt_payload = json.dumps(batch)  # Преобразование данных в JSON строку
            mqtt_client.publish(mqtt_topic, mqtt_payload)
            print(f"Published batch {i // batch_size + 1} to MQTT topic '{mqtt_topic}'")
            time.sleep(1)  # Задержка между публикациями

    except FileNotFoundError:
        print(f"File '{filename}' not found")
    except json.JSONDecodeError as e:
        print(f"Error decoding JSON from file '{filename}': {str(e)}")
    except Exception as e:
        print(f"Error publishing data to MQTT: {str(e)}")
```

```python
def on_connect(mqtt_client, userdata, flags, rc):
    print(f"Connected with result code {rc}")
    mqtt_client.subscribe("sensors/devices")  # Подписка на топик "sensors/restaurants"

# Callback функция при получении сообщения
def on_message(mqtt_client, userdata, msg):
    print(f"Received message: {msg.topic} {msg.payload.decode()}")
    try:
        batch = json.loads(msg.payload.decode())  # Декодируем JSON-полезную нагрузку
        for document in batch:
            process_and_store_data(document)
    except json.JSONDecodeError:
        print("Failed to decode JSON")
```

The on_connect and on_message functions are callback functions used to process events when working with an MQTT client. They are implemented using the paho.mqtt.client library to connect to an MQTT server and receive messages from topics.

Storing in MongoDB

```python
def store_in_mongodb(document):
    mongo_document = {
        "device_id": document["device_id"],
        "timestamp": document["timestamp"],
        "motion" : document["motion"],
        "latitude" :document["location"]["latitude"],
        "longitude" :document["location"]["longitude"]
    }
    collection.insert_one(mongo_document)
    print(f"Data stored in MongoDB: {mongo_document}")
```

Storing in Neo4j

```python
def store_in_neo4j(document):
    device_id = document['device_id']

    # Создание узла устройства
    neo4j_graph.run(
        """
        MERGE (d:Device {id: $device_id})
        """,
        device_id=device_id
    )

    # Проверяем наличие поля data и data_type в документе
    if 'data' in document and 'data_type' in document['data']:
        data_type = document['data']['data_type']['type']
    else:
        data_type = 'Unknown'

    # Создание узла типа данных
    neo4j_graph.run(
        """
        MERGE (t:DataType {type: $data_type})
        """,
        data_type=data_type
    )

    # Создание отношения HAS_DATA_TYPE между устройством и типом данных
    neo4j_graph.run(
        """
        MATCH (d:Device {id: $device_id})
        MATCH (t:DataType {type: $data_type})
        MERGE (d)-[:HAS_DATA_TYPE]->(t)
        """,
        device_id=device_id, data_type=data_type
    )

    print(f"Data stored in Neo4j: Device '{device_id}' with data type '{data_type}'")
```

Storing in MySQL

```python
def store_in_mysql(document):
    mysql_conn = connect_to_mysql()
    if mysql_conn is None:
        print("Failed to connect to MySQL")
        return
    mysql_cursor = mysql_conn.cursor()

    # Insert data into Devices table
    device_query = """
    INSERT INTO Devices (device_id, timestamp, motion, latitude, longitude)
    VALUES (%s, %s, %s, %s, %s)
    ON DUPLICATE KEY UPDATE
        timestamp=VALUES(timestamp),
        motion=VALUES(motion),
        latitude=VALUES(latitude),
        longitude=VALUES(longitude)
    """
    device_values = (
        document['device_id'],
        document['timestamp'],
        document['motion'],
        document['location']['latitude'],
        document['location']['longitude']
    )
    mysql_cursor.execute(device_query, device_values)

    # Insert data into DataTypes table
    data_type_query = """
    INSERT INTO DataTypes (type, ssid, signal_strength)
    VALUES (%s, %s, %s)
    ON DUPLICATE KEY UPDATE
        ssid=VALUES(ssid),
        signal_strength=VALUES(signal_strength)
    """
    data_type_values = (
        document['data']['data_type']['type'],
        document['data']['data_type']['ssid'],
        document['data']['data_type']['signal_strength']
    )
    mysql_cursor.execute(data_type_query, data_type_values)

    # Insert data into DeviceDataTypes table
    device_data_type_query = """
    INSERT INTO DeviceDataTypes (device_id, data_type)
    VALUES (%s, %s)
    ON DUPLICATE KEY UPDATE
        device_id=VALUES(device_id),
        data_type=VALUES(data_type)
    """
    device_data_type_values = (
        document['device_id'],
        document['data']['data_type']['type']
    )
    mysql_cursor.execute(device_data_type_query, device_data_type_values)

    mysql_conn.commit()
    print(f"Data stored in MySQL: {document}")
```

MySQL database

DataTypes	Browse	Structure	Search	Insert	Empty	Drop	3	InnoDB	utf8mb4_general_ci	16.0 KiB
DeviceDataTypes	Browse	Structure	Search	Insert	Empty	Drop	90	InnoDB	utf8mb4_general_ci	32.0 KiB
Devices	Browse	Structure	Search	Insert	Empty	Drop	66	InnoDB	utf8mb4_general_ci	16.0 KiB

Data types table

Edit	Copy	Delete	Bluetooth	Bluetooth_Device2	-62
Edit	Copy	Delete	WiFi	Home_Network	-90
Edit	Copy	Delete	Zigbee	Zigbee_Device3	-17

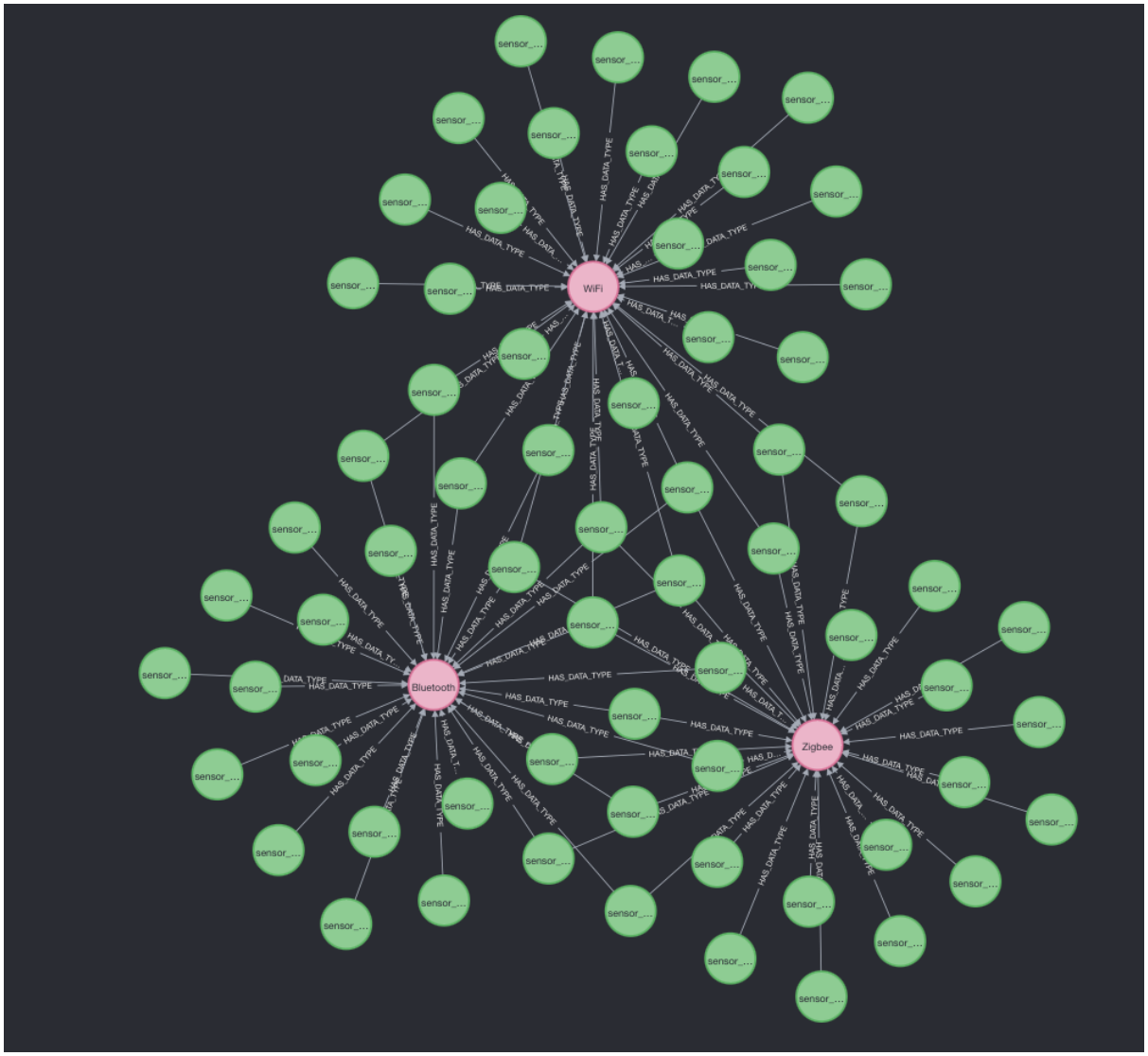
DeviceDataTypes table

Edit	Copy	Delete	sensor_001	Bluetooth
Edit	Copy	Delete	sensor_001	WiFi
Edit	Copy	Delete	sensor_001	Zigbee
Edit	Copy	Delete	sensor_002	WiFi
Edit	Copy	Delete	sensor_003	Zigbee
Edit	Copy	Delete	sensor_004	Bluetooth
Edit	Copy	Delete	sensor_005	Bluetooth
Edit	Copy	Delete	sensor_005	Zigbee
Edit	Copy	Delete	sensor_008	WiFi
Edit	Copy	Delete	sensor_009	Bluetooth
Edit	Copy	Delete	sensor_009	WiFi
Edit	Copy	Delete	sensor_009	Zigbee
Edit	Copy	Delete	sensor_010	Bluetooth
Edit	Copy	Delete	sensor_012	WiFi
Edit	Copy	Delete	sensor_013	Bluetooth
Edit	Copy	Delete	sensor_014	Bluetooth
Edit	Copy	Delete	sensor_014	Zigbee
Edit	Copy	Delete	sensor_015	Bluetooth

Devices table

☐	Edit	Copy	Delete	sensor_001	2024-06-17 12:12:12	0	-27.913838	-19.714138
☐	Edit	Copy	Delete	sensor_002	2024-06-17 11:27:07	0	33.451109	-109.028338
☐	Edit	Copy	Delete	sensor_003	2024-06-17 11:13:32	0	11.964254	99.428537
☐	Edit	Copy	Delete	sensor_004	2024-06-17 11:15:19	0	32.604991	-132.992578
☐	Edit	Copy	Delete	sensor_005	2024-06-17 11:40:25	1	-39.026975	114.901127
☐	Edit	Copy	Delete	sensor_008	2024-06-17 11:27:12	0	-54.703950	50.840583
☐	Edit	Copy	Delete	sensor_009	2024-06-17 11:37:48	0	-24.867076	124.062490
☐	Edit	Copy	Delete	sensor_010	2024-06-17 12:00:18	1	-51.590464	113.771108
☐	Edit	Copy	Delete	sensor_012	2024-06-17 11:51:01	0	-28.482050	116.127018

Neo4j database



MongoDB database


```
_id: ObjectId('667039653ff337531fcdc9e6')
device_id : "sensor_004"
timestamp : "2024-06-17T11:15:19.203155"
motion : false
latitude : 32.604991
longitude : -132.992578
```

```
_id: ObjectId('6670397baa2582b598b12e7c')
device_id : "sensor_004"
timestamp : "2024-06-17T11:15:19.203155"
motion : false
latitude : 32.604991
longitude : -132.992578
```

```
_id: ObjectId('66703a09774712abff996129')
device_id : "sensor_004"
timestamp : "2024-06-17T11:15:19.203155"
motion : false
latitude : 32.604991
longitude : -132.992578
```