Do Arduino ao Banco de Dados SQL ou NoSQL

Fernando Antonio Fernandes Anselmo - Versão 2.0

Introdução

Nesta apostila vamos aprender como obter dados de sensores do mundo real, mas especificamente um sensor de Luminosidade juntamente com de temperatura, esse projeto possui um custo extremamente reduzido e o único componente "com um custo mais elevado"seria o **Arduino Uno** que pode ser reduzido com uma substituição por um **Arduino Micro** (que possui o custo bem mais acessível). Porém a compra do Uno é uma boa aquisição para qualquer Maker que se interesse por prototipação de projetos eletrônicos.

Como fonte de dados utilizaremos os cinco bancos a seguir:

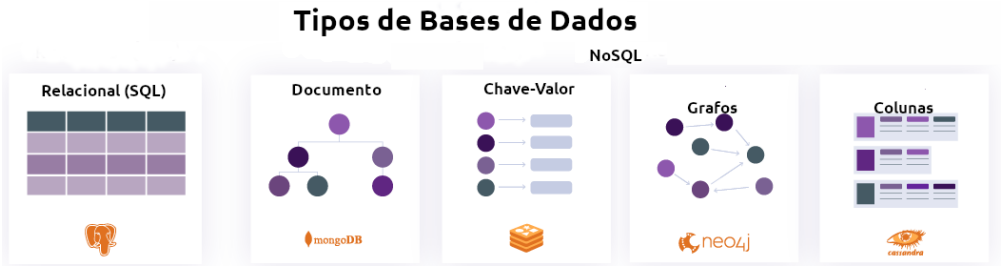

Figura 1: *Bancos de Dados utilizados*

Utilizaremos no padrão SQL os SGBDR mais conhecidos do mercado **MySQL** e **Postgres**, além disso, alternativamente quatro NoSQL: **MongoDB** representando o tipo orientado a Documento, **Redis** orientado a Chave-Valor, **Neo4j** orientado a Grafo e **Apache Cassandra** orientado a Coluna, vai ser uma experiência interessante e comparativa pois podemos assim definir qual desses se encaixa melhor em um projeto que pretendemos realizar. Lembre-se apenas que ideia não é julgarmos, mas como característica comparativa e de aprendizado, pois cada banco de dados possui suas forças e fraquezas.

Para realizar o trabalho de intermediário (ou seja) quem lerá os dados da serial e enviará para o banco de dados será a linguagem Python. Essa é conhecida por sua simplicidade e facilidade de uso. Usaremos a porta serial como meio de obtermos os dados, assim é necessário instalar a biblioteca **PySerial**:

```
$ pip install pyserial
```

E o padrão JSON para essa passagem, devemos também instalar a biblioteca **SimpleJson**:

```
$ pip install simplejson
```

Montagem do Projeto

Começamos com a montagem do projeto. Como entrada de dados usamos um componente chamado de *Photoresistor* ou simplesmente **LDR** (abreviatura para *Light Dependent Resistor*), isso é uma fotocélula e funciona como um resistor que varia com a intensidade de luz, quanto mais luz, mais alta é a resistência. Em conjunto um sensor de temperatura que também é conhecido como **LM35**.

O preço de ambos não passa de poucos reais em qualquer casa de eletrônica.

Lista de componentes do projeto:

- 1 Arduino Uno Rev 3
- 1 LDR
- 1 LM35
- 1 Resistor de 1 KΩ
- 6 jumpers para as conexões
- 1 micro protoboard (ou qualquer outra similar)

As conexões são realizadas conforme a seguinte figura:

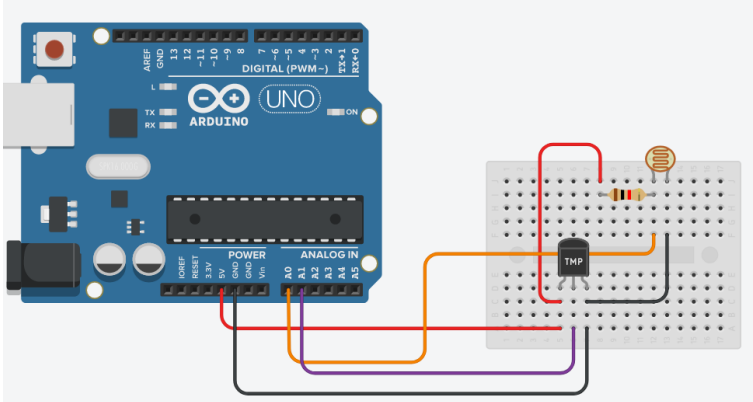

Figura 2: *Circuito da Ligação*

Em uma de suas pernas do **LDR**, ligamos um *jumper* para fazermos uma ponte para a GND do **LM35**, a outra perna possui uma dupla ligação: um *jumper* para porta analógica **A0** e um resistor de 1 KΩ ligado em 5V como outra ponte para a alimentação do **LM35**.

O componente **LM35** possui 3 pernas, para saber quais são o lado achatado desse sensor deve estar virado para nossa frente: a perna mais esquerda é ligada por um *jumper* a alimentação de 5V, a central um *jumper* para a porta analógica **A1** e a perna mais direita um *jumper* para a **GND**.

Para a programação no Arduino, faremos a seguinte codificação:

```
#define LDR A0
#define LM35 A1

int lums = 0;
float tensao;
float grauC;
float grauF;

void setup() {
  pinMode(LDR, INPUT);
  pinMode(LM35, INPUT);
  Serial.begin(9600);
}

void loop() {
  lums = map(analogRead(LDR), 244, 1100, 0, 100);
  tensao = (float(analogRead(LM35))*5)/1023; // 5V e leitura analógica 0 a 1023
  grauC = tensao / 0.010; // 0,010 mV
```

```
19    grauF = grauC * (9.0/5.0) + 32.0;
20    Serial.println("{\"lums\":" + String(lums) + ", \"grauC\":"
21     + String(grauC) + ", \"grauF\":" + String(grauF) + "}");
22    delay(5000); // 5 segundos
23  }
```

Definimos uma constante **LDR** que aponta para a porta **A0** e outra **LM35** a porta **A1**. Definimos uma variável inteira chamada **lums** e três flutuantes que respectivamente conterão o valor da tensão, graus em Celsius e Fahrenheit. No método *setup()* definimos as portas dos sensores como saída e iniciamos a serial na velocidade de **9.600 bounds** (isso é muito importante).

No método loop() obtemos a informação do componente que varia entre 244 a 1.100, porém para ficar mais "harmônico"criamos um mapa de valores que converte esse intervalo entre 0 a 100. Para obtermos o valor da temperatura primeiro calculamos a tensão, este sensor trabalha a 5V em uma porta analógica que varia de 0 a 1023. Basta agora dividir a informação por 0,010 que teremos o valor em graus *Celsius*. E a partir dessa aplicamos uma fórmula de conversão para obtermos a variação em graus *Fahrenheit*.

O pulo do gato é montarmos a informação de saída na serial em formato JSON (*JavaScript Object Notation*) que é um modelo para armazenamento e transmissão de informações no formato texto. É bastante utilizado por aplicações diversas devido sua capacidade em estruturar informações de modo compacto é prático, este formato será importante para o programa que recebe os dados. Esperamos por 5 segundos para a próxima leitura.

Ao ativarmos a Serial Monitor obtemos as seguintes respostas:

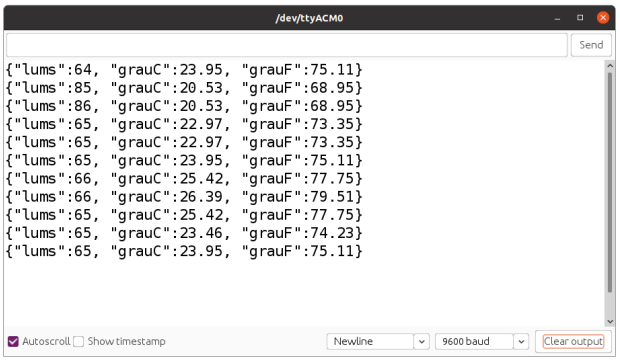

Figura 3: *Saída da Serial*

NOTA. Aqui estamos usando o tempo de 5 segundos apenas como característica de teste, mas na prática podemos aumentar para minutos pois não existe uma diferença muito grande de temperatura ou luminosidade em um intervalo de tempo em segundos. Qual o motivo de usamos um sensor de luminosidade juntamente como um de temperatura? Existe uma relação entre essas grandezas, e esse é o ponto de partida para uma interessante pesquisa podemos tentar explorar esse assunto em futuros trabalhos.

Banco de dados padrão SQL

Esse tipo de Banco trabalha com SQL (Linguagem de Consulta Estruturada, do inglês *Structured Query Language*) como interface. Normalmente Bancos de dados nesse padrão são conhecidos como SGBDR - Sistema de Gerenciamento de Banco de Dados Relacional (ou RDBMS do inglês

Relational Database Manager System).

MySQL

O primeiro banco de dados que utilizaremos é o MySQL. **MySQL** é um SGBDR gratuito e de código aberto. Seu nome é uma combinação de "My", o nome da filha do co-fundador Michael Widenius, e "SQL", a abreviação de *Structured Query Language*. É atualmente um dos bancos mais populares, e atualmente pertence a *Oracle Corporation*, com mais de 10 milhões de instalações pelo mundo.

Após estar instalado corretamente em seu ambiente. Criamos um database:
`create database projetoint`

Este tipo de SGBDR utiliza databases para separar as tabelas e permitir uma melhor organização. Usamos esse database:
`use projetoint`

E criamos a seguinte entidade:

```
CREATE TABLE dados (
  datahora timestamp NOT NULL,
  ldr int4 NOT NULL,
  tempC float8 NOT NULL,
  tempF float8 NOT NULL,
  CONSTRAINT ldr_pkey PRIMARY KEY (datahora)
);
```

Nossa entidade possui os campos: **datahora** como chave primária que contém a data e hora da coleta, **ldr** com o valor do sensor de luminosidade em uma escala de 0 a 100, **tempC** para o valor do sensor da temperatura em graus na escala Celsius e **tempF** para o valor do sensor da temperatura em graus na escala Farenheit.

Para que o Python possa se comunicar com o MySQL é necessário instalar a biblioteca **mysql-connector-python**:
`$ python -m pip install mysql-connector-python`

Importamos as bibliotecas necessárias:

```
import mysql.connector
import serial
import simplejson
from datetime import datetime
```

Criamos o seguinte método para gravar os dados no banco de dados:

```
def gravar_dados(conn, JSON):
  try:
    cursor = con.cursor()
    cursor.execute("""
      INSERT INTO dados (datahora, ldr, tempC, tempF)
      VALUES (%(data)s, %(lums)s, %(grauC)s, %(grauF)s);
      """, JSON)
    con.commit()
    count = cursor.rowcount
```

```
    print(count, "Registro inserido com sucesso!")
  except (Exception) as error:
    print("Falhou para inserir registro!", error)
```

Essa função recebe a conexão com o banco e dado em formato JSON lido. Criamos um objeto cursor (a conversação com o MySQL é realizada através desse objeto). Executamos um **INSERT** no banco que monta os dados através do JSON passado. Damos um *commit()* para gravar os dados, obtemos a resposta quantas linhas foram executadas com o comando, mostramos esse resultado e encerramos a conexão. Caso ocorra qualquer problema este será mostrado um erro pois existe uma proteção do código com um comando *try-except*.

E o seguinte método para obter os dados do componente:

```
def ler_dados_serial():
  ser = serial.Serial('/dev/ttyACM0', 9600, timeout=5)
  con = mysql.connector.connect(
      host="localhost",
      user="root",
      password="root",
      database="projetoint"
  )
  while True:
    try:
      ler = ser.readline()
      json = simplejson.loads(ler.decode('utf8').replace("'", '"'))
    except (Exception) as error:
      print("Falhou para ler registro!", error)
      continue
    dt_str = datetime.now().strftime("%Y/%m/%d %H:%M:%S")
    json["data"] = dt_str
    gravar_dados(con, json)

  con.close()
```

Iniciamos a variável **ser** com os dados da serial (lembrar que ao rodar esse programa, a janela **Serial Monitor** da Arduino IDE deve estar fechada para evitarmos conflito), são três indicações: a porta que o Arduino está ativo, a velocidade de leitura (que acertamos a 9.600 bounds) e o tempo que deve ser realizada a leitura (cada 5 segundos).

Para a conexão criamos uma variável **con** que contém os seguintes valores:

- **host** - servidor (IP) que se encontra o banco.
- **user** - o nome do usuário.
- **password** - senha desse usuário.
- **database** - o nome da base de dados.

Entramos em um laço que colherá os dados. Neste lemos o conteúdo da serial e o colocamos na variável **ler**, usamos um objeto da biblioteca *simplejson* para converter seu conteúdo para o padrão JSON correto. Para obter os dados decodificamos a saída para o padrão UTF-8 (muitos sistemas Windows exigem essa cláusula, já no padrão Unix não existe essa necessidade).

Adicionamos o campo **data** com o valor da data e hora atual (outro pulo do gato) sempre que for gravar data em um banco de dados use o formato **Y/M/D**. Os formatos **D/M/Y** e **M/D/Y** dependem de como está configurado o banco, já o formato **Y/M/D** não tem erro pois quando o primeiro valor possui 4 dígitos e o banco assume esse formato.

Gravamos esse na base com uma chamada ao método anteriormente visto com a passagem da variável de conexão e o JSON da leitura. E agora basta disparar esse método na chamada principal:

```
if __name__ == '__main__':
  ler_dados_serial()
```

E podemos armazenar os dados de luminosidade e temperatura para um determinado ambiente. Outra dica interessante seria trocar o componente **LM35** (que mede apenas temperatura) por um sensor denominado **DHT11** (que além da temperatura mede também a umidade).

Postgres

PostgreSQL, também conhecido como Postgres, é um SGBD (Sistema Gerenciador de Banco de Dados) relacional gratuito e de código aberto que enfatiza a extensibilidade e conformidade com padrão SQL. Seu nome refere-se às suas origens como sucessor do banco **Ingres** e desenvolvido na Universidade da Califórnia, em *Berkeley*. Este deve estar instalado corretamente em seu ambiente. Por padrão do Postgres, estaremos alocados em um Database de mesmo nome.

Neste criamos um esquema:
`create schema projetoint`

Sempre que trabalhamos com bancos de dados que assim permitam é preferível usar um esquema para separar as tabelas e permitir uma melhor organização. E a seguinte entidade:

```
CREATE TABLE projetoint.dados (
  datahora timestamp NOT NULL,
  ldr int4 NOT NULL,
  tempC float8 NOT NULL,
  tempF float8 NOT NULL,
  CONSTRAINT ldr_pkey PRIMARY KEY (datahora)
);
```

A entidade criada é a cópia exata como vimos para o MySQL. Para que o Python possa se comunicar com o Postgres é necessário instalar a biblioteca **psycopg2**:
`$ pip install psycopg2-binary`

Importamos as bibliotecas necessárias:

```
import psycopg2
import serial
import simplejson
from datetime import datetime
```

Criamos o seguinte método para gravar os dados no banco de dados:

```
def gravar_dados(con, JSON):
  try:
    cursor = con.cursor()
    cursor.execute("""
      INSERT INTO projetoint.dados (datahora, ldr, tempC, tempF)
      VALUES (%(data)s, %(lums)s, %(grauC)s, %(grauF)s);
      """, JSON)
    con.commit()
```

```
    count = cursor.rowcount
    print(count, "Registro inserido com sucesso!")
  except (Exception) as error:
    print("Falhou para inserir registro!", error)
```

Comparando com a função já vista para o banco MySQL apenas o comando INSERT foi modificado para adicionarmos o esquema do banco. E o seguinte método para obter os dados do componente:

```
def ler_dados_serial():
  ser = serial.Serial('/dev/ttyACM0', 9600, timeout=5)
  con = psycopg2.connect(
      "host=localhost port=5432 dbname=postgres user=postgres password=postgres")
  while True:
    try:
      ler = ser.readline()
      json = simplejson.loads(ler.decode('utf8').replace("'", '"'))
    except (Exception) as error:
      print("Falhou para ler registro!", error)
      continue
    dt_str = datetime.now().strftime("%Y/%m/%d %H:%M:%S")
    json["data"] = dt_str
    gravar_dados(con, json)

  con.close()
```

Novamente pouquíssimas modificações aqui somente quanto ao objeto de conexão **con** que contém os seguintes valores:

- **host** - servidor (IP) que se encontra o banco.
- **port** - porta que se encontra o banco.
- **dbname** - o nome da base de dados.
- **user** - o nome do usuário.
- **password** - senha desse usuário.

Gravamos esse na base com uma chamada ao método anteriormente visto com a passagem da variável de conexão e o JSON da leitura. E agora basta disparar esse método na chamada principal:

```
if __name__ == '__main__':
  ler_dados_serial()
```

O que percebemos é todo banco SQL processa a inclusão dos valores praticamente de mesmo modo, sendo necessário mudar o conector e adaptar a instrução SQL de forma correta.

Bancos de Dados NoSQL

Existem quatro tipos de bancos no padrão NoSQL, não pretendo aqui debater a vantagem de cada um deles, vamos realizar as modificações necessárias no programa em Python possa se utilizar um de cada tipo.

MongoDB - Tipo: Orientado a Documento

Para gravarmos os dados necessitamos da biblioteca "pymongo"que pode ser instalada com o comando:
`$ pip install pymongo`

Importamos essa biblioteca para o programa:

```
from pymongo import MongoClient
```

E no método *gravar_dados()* modificar para a seguinte codificação:

```
def gravar_dados(json):
  cliente = MongoClient("localhost", 27017)
  db = cliente['coleta']
  col = db['dados']
  chv = col.insert_one(json)
  print("Registro Inserido: ", chv.inserted_id)
  cliente.close()
```

Realizamos a conexão com o banco de dados que deve estar disponível na porta 27017, criamos ou abrimos a base "coleta"e a coleção "dados"e inserimos o registro do JSON enviado através do comando **insert_one**, essas três linhas poderiam ser reescritas para:
`cliente['coleta']['dados'].insert_one(json)`

Porém para facilitar o entendimento deixamos com três objetos. Mostramos a chave do registro inserido e encerramos a conexão.

Redis - Tipo: Orientado a Chave-Valor

Para gravarmos os dados necessitamos da biblioteca "redis"que pode ser instalada com o comando:
`$ pip install redis`

Importamos essa biblioteca para o programa:

```
import redis
```

E no método *gravar_dados()* modificar para a seguinte codificação:

```
def gravar_dados(json):
  r = redis.Redis(host='localhost', port='6379')
  with r.pipeline() as pipe:
    chave = json["data"]
    pipe.hset(chave, "ldr", json["lums"])
    pipe.hset(chave, "tempC", json["grauC"])
    pipe.hset(chave, "tempF", json["grauF"])
    pipe.execute()
  r.bgsave()
  r.close()
```

De modo extremamente simples, criamos uma conexão com o banco, e usamos a técnica de pipeline para que possamos gravar um registro chave-valor com subcampos chave-valor. O dado de **datahora**

será a chave principal do registro (pois esse nunca se repete), e gravamos os outros 3 campos obtendo os dados do arquivo JSON e enviando-os para o padrão chave-valor do Redis.

Neo4j - Tipo: Orientado a Grafo

Para gravarmos os dados necessitamos da biblioteca "neo4j"que pode ser instalada com o comando:
`$ pip install neo4j`

Importamos essa biblioteca para o programa:

```
from neo4j import GraphDatabase
```

Adicionamos uma constante que vai manter nosso comando *Cypher* para criar o registro, abaixo das importações de bibliotecas:

```
cqlCriar = "CREATE (:coleta { data: $data, ldr: $ldr, tempC: $tempC, tempF: $tempF})"
```

E modificamos o método *gravar_dados()*:

```
  def gravar_dados(json):
    graphDB_Driver = GraphDatabase.driver(
      "bolt://localhost:7687", auth=("neo4j", "test"))
    with graphDB_Driver.session() as secao:
      secao.run(cqlCriar, data=json["data"],
        ldr=json["lums"],
        tempC=json["grauC"],
        tempF=json["grauF"])
      secao.close()
```

Criamos uma conexão com o banco, e abrimos uma seção para essa conexão. Em seguida executamos o comando *Cypher* e adicionamos a cada parâmetro (iniciado por `$`) o seu respectivo valor obtido através do JSON. E encerramos a seção aberta.

Apache Cassandra - Tipo: Orientado a Coluna

Antes de qualquer atividade no Python precisamos acessar o **CQL** do banco Apache Cassandra para digitarmos os seguintes comandos para criarmos *KeySpace*:
`cqlsh> CREATE KEYSPACE coleta WITH replication = {'class':'SimpleStrategy', 'replication_factor':1};`

E a tabela que receberá os dados:
`cqlsh> use coleta;`
`cqlsh:coleta> CREATE TABLE dados (datahora timestamp PRIMARY KEY, ldr int, tempC float, tempF float);`

Para gravarmos os dados necessitamos da biblioteca "cassandra-driver"que pode ser instalada com o comando:
`$ pip install cassandra-driver`

Através dessa biblioteca importamos 2 objetos que utilizaremos:

```
from cassandra.cluster import Cluster
```

```
from cassandra.query import SimpleStatement
```

Adicionamos novos métodos, pois a coluna *timestamp* trabalha com elementos do tipo inteiro longo:

```
def unix_time(dt):
  epoch = datetime.utcfromtimestamp(0)
  delta = dt - epoch
  return delta.total_seconds()

def unix_time_millis(dt):
  return int(unix_time(dt) * 1000.0)
```

No método *ler_dados_serial()* adicionamos os objetos para nos conectarmos ao Cassandra:

```
def ler_dados_serial():
 ser = serial.Serial('/dev/ttyACM0', 9600, timeout=4)
 session = Cluster().connect()
 session.execute("USE coleta;")
```

Passamos este na chamada ao método *gravar_dados()*:

```
  gravar_dados(session, json)
```

E na montagem do elemento *data* para o JSON, realizamos a seguinte modificação:

```
  dt_long = unix_time_millis(datetime.now())
  json["data"] = dt_long
```

E modificamos o método *gravar_dados()*:

```
def gravar_dados(session, JSON):
  try:
    session.execute("""
      INSERT INTO dados (datahora, ldr, tempC, tempF)
      VALUES (%(data)s, %(lums)s, %(grauC)s, %(grauF)s);""", JSON)
    print("Registro inserido com sucesso!")
  except (Exception) as error:
    print("Falhou para inserir registro!", error)
```

E agora podemos escolher qual base de dados seja padrão SQL ou NoSQL se encaixa melhor para o projeto que estamos realizando.