02_MySQL

March 13, 2020

1 Acesso a bases de dados MySQL com Python

Pedro Cardoso
(ISE/UAlg - pcardoso@ualg.pt)

1.1 Estabelecimento de conexão à base de dados usando um Connector/Python

o método `connect()` cria uma conexão a um servidor MySQL e devolve um objeto `MySQLConnection`. (ver https://dev.mysql.com/doc/connector-python/en/connector-python-connectargs.html para outros argumentos e opções)

```
In [1]: import mysql.connector

        cnx = mysql.connector.connect(user='sensors',
                                      password='##sensors##',
                                      host='localhost', # replace 'localhost', if necessary
                                      database='sensors')
        cnx
```

```
Out[1]: <mysql.connector.connection_cext.CMySQLConnection at 0x7f76903e8a20>
```

Algumas informações sobre a conexão podem ser consultadas no `__dict__`

```
In [2]: cnx.__dict__
```

```
Out[2]: {'_cmysql': <_mysql_connector.MySQL at 0x17dceb0>,
         '_columns': [],
         'converter': None,
         '_client_flags': 1286669,
         '_charset_id': 45,
         '_sql_mode': None,
         '_time_zone': None,
         '_autocommit': False,
         '_server_version': (5, 5, 5),
         '_handshake': {'protocol': 10,
          'server_version_original': '5.5.5-10.4.11-MariaDB',
          'server_threadid': 139,
          'charset': None,
```

```
 'server_status': None,
 'auth_plugin': None,
 'auth_data': None,
 'capabilities': -2113931266},
'_conn_attrs': {'_connector_name': 'mysql-connector-python',
 '_connector_license': 'GPL-2.0',
 '_connector_version': '8.0.19',
 '_source_host': 'pcardoso-kubuntu-desktop'},
'_user': 'sensors',
'_password': '##sensors##',
'_database': 'sensors',
'_host': 'localhost',
'_port': 3306,
'_unix_socket': None,
'_client_host': '',
'_client_port': 0,
'_ssl': {},
'_ssl_disabled': False,
'_force_ipv6': False,
'_use_unicode': True,
'_get_warnings': False,
'_raise_on_warnings': False,
'_connection_timeout': None,
'_buffered': False,
'_unread_result': False,
'_have_next_result': False,
'_raw': False,
'_in_transaction': False,
'_prepared_statements': None,
'_ssl_active': False,
'_auth_plugin': '',
'_pool_config_version': None,
'_converter_class': None,
'_compress': False,
'_consume_results': False}
```

E no final devemos libertar sempre a conexão

```
In [3]: cnx.close()
```

1.1.1 Ficheiro de configuração e tratamento de exceções

De um modo geral é aconselhável * fazer tratamento de exceções * e criar um ficheiro de configuração (config.py)

```
config = {
    'host' : 'localhost',
    'user' : 'sensors',
    'password' : '##sensors##',
```

```
    'db' : 'sensors'
}
```

e depois fazer...

```
In [4]: # Comecamos por importar o ficheiro de configuração

        # Se correr a partir de um script python "normal",  poderá depender do sistema e faz-s
        # from config import config
        # from .config import config

        # em Jupyter fazemos
        %run config.py

        import mysql.connector

        try:
            cnx = mysql.connector.connect(**config)
        except mysql.connector.Error as err:
            print('Ups! Ocorreu um erro!')
            print(err)
        else:
            print('Sucesso!')
            cnx.close()
```

```
Sucesso!
```

1.1.2 Exercício

1. Experimentem a desligar o servidor e correr a linha acima: qual a mensagem de erro?
2. Mude o nome do utilizador no ficheiro de configuração e correr a linha acima: qual a mensagem de erro?
3. Tratem as exceções de modo adequatado.

1.2 Operações de DDL: Criação de uma base de dados

Para a criação das tabelas e relacionamentos podemos construiro o sql ou, como alternativa, podemos usar ferramentas como sejam o MySQL Workbench, o Phpmyadmin, o SQlite Browser, o DataGrip, etc.

Consideremos o caso em que contruímos o sql...

Comecemos por criar uma base de dados no servidor de MySQL (façam sempre tratamento de exceções...).

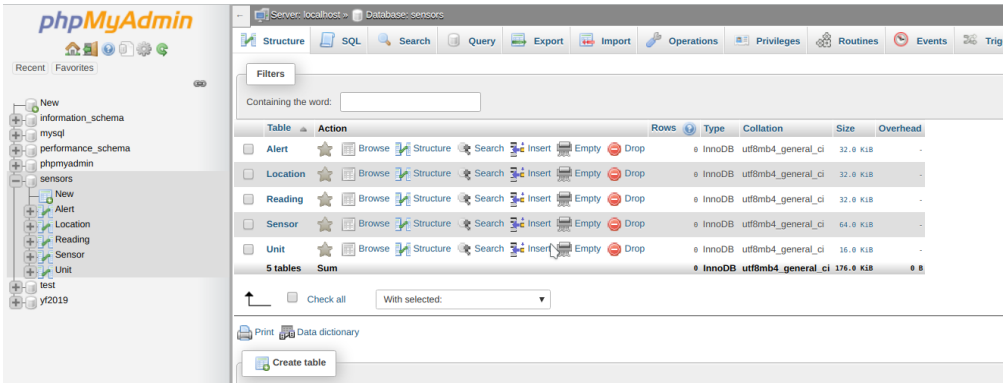
```
In [5]: # from config import config as conf
        %run config.py
        import mysql.connector

        sql = '''
```

```
CREATE SCHEMA IF NOT EXISTS `sensors` DEFAULT CHARACTER SET utf8 ;

USE `sensors` ;

-- -----------------------------------------------------
-- Table `sensors`.`Location`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `sensors`.`Location` (
  `idLocation` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(45) NOT NULL,
  `description` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`idLocation`),
  UNIQUE INDEX `name_UNIQUE` (`name` ASC)) ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `sensors`.`Unit`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `sensors`.`Unit` (
  `unit` VARCHAR(45) NOT NULL,
  `description` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`unit`)) ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `sensors`.`Sensor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `sensors`.`Sensor` (
  `idSensor` INT NOT NULL AUTO_INCREMENT,
  `idLocation` INT NOT NULL,
  `name` VARCHAR(45) NOT NULL,
  `unit` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`idSensor`),
  INDEX `fk_Sensor_Location_idx` (`idLocation` ASC),
  INDEX `fk_Sensor_Units1_idx` (`unit` ASC),
  UNIQUE INDEX `uniq_loc_vs_sensor` (`idLocation` ASC, `name` ASC),
  CONSTRAINT `fk_Sensor_Location`
    FOREIGN KEY (`idLocation`)
    REFERENCES `sensors`.`Location` (`idLocation`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Sensor_Units1`
    FOREIGN KEY (`unit`)
    REFERENCES `sensors`.`Unit` (`unit`)
    ON DELETE CASCADE
    ON UPDATE CASCADE) ENGINE = InnoDB;
```

```
    -- -----------------------------------------------------
    -- Table `sensors`.`Reading`
    -- -----------------------------------------------------
    CREATE TABLE IF NOT EXISTS `sensors`.`Reading` (
      `idReading` INT NOT NULL AUTO_INCREMENT,
      `idSensor` INT NOT NULL,
      `timestamp` TIMESTAMP NOT NULL,
      `value` FLOAT NOT NULL,
      PRIMARY KEY (`idReading`),
      INDEX `fk_Reading_Sensor1_idx` (`idSensor` ASC),
      CONSTRAINT `fk_Reading_Sensor1`
        FOREIGN KEY (`idSensor`)
        REFERENCES `sensors`.`Sensor` (`idSensor`)
        ON DELETE CASCADE
        ON UPDATE CASCADE) ENGINE = InnoDB;


    -- -----------------------------------------------------
    -- Table `sensors`.`Alert`
    -- -----------------------------------------------------
    CREATE TABLE IF NOT EXISTS `sensors`.`Alert` (
      `idAlert` INT NOT NULL AUTO_INCREMENT,
      `idSensor` INT NOT NULL,
      `timestamp` TIMESTAMP NOT NULL,
      `description` VARCHAR(45) NOT NULL,
      `cleared` BIT NULL,
      PRIMARY KEY (`idAlert`),
      INDEX `fk_Alert_Sensor1_idx` (`idSensor` ASC),
      CONSTRAINT `fk_Alert_Sensor1`
        FOREIGN KEY (`idSensor`)
        REFERENCES `sensors`.`Sensor` (`idSensor`)
        ON DELETE CASCADE
        ON UPDATE CASCADE) ENGINE = InnoDB;
'''
```

Depois de termos o SQL/DDL fazemos

```
In [6]: try:
            # Abrimos a conexão
            cnx = mysql.connector.connect(**config)
            # criamos um cursor a partir da conexão
            cursor = cnx.cursor()
            # executamos o query sql
            cursor.execute(sql)
        except mysql.connector.Error as err:
            print(err)
        else:
            print('ok!')
            cnx.close()
```



phpmyadmin_database_created.png

ok!

O comando `cnx.cursor()` devolve um objeto da classe `MySQLCursor` que podem executar operações como instruções SQL. Objetos de `cursor` interagem com o servidor MySQL usando um objeto `MySQLConnection`. Para mais informações ver https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlcursor.html

No phpmyadmin devem ver algo como

1.3 Operações CRUD

1.3.1 `INSERT`

Aberta a conexão em MySQL

```
In [7]: cnx = mysql.connector.connect(**config)
        cursor = cnx.cursor()
```

localização Uma boa estratégia é definir variáveis no sql usando parametros no estilo `%s` or `%(nome)s` (i.e., usar o estilo "format" ou "pyformat") e umj tuplo com os dados

```
In [8]: sql = '''
            INSERT INTO Location
                (idLocation, name, description)
            VALUES
                (DEFAULT, %s, %s)
        '''
        # e um tuplo com os dados
        data = ('Prometheus Server', 'Prometheus Server @ lab. 163 / ISE /UAlg')
```

inserir uma nova localização na base de dados e obter o id correspondente, guardado em `location_id` e que iremos usar à frente

```
In [9]: cursor.execute(sql, data)
        location_id = cursor.lastrowid
        location_id
```

```
Out[9]: 1
```

Quando estamos a usar um sistema transacional, como o InnoDB, temos de efetuar o "commit" depois de fazer um INSERT, DELETE, ou UPDATE (comandos que alterem tabelas). ver (https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlconnection-commit.html)

```
In [10]: cnx.commit()
```

Unit Inserir uma nova `Unit`, se não existir (ver a documentação do comando `REPLACE`)

```
In [11]: sql = '''
             REPLACE INTO Unit
                 (unit, description)
             VALUES
                 ("percent", "percentage of usage")
         '''

         cursor.execute(sql)
         cnx.commit()
```

Sensor Inserir um novo sensor e obter o seu id, preparando o sql

```
In [12]: # prepare o sql
         sql = '''INSERT INTO Sensor (idSensor, idLocation, name, unit)
                 VALUES (DEFAULT, %(idLocation)s, %(name)s, %(unit)s);'''
```

Preparar os dados, agora com um dicionário que tem em conta `%(idLocation)s, %(name)s, %(unit)s`

```
In [13]: data = {
                  'idLocation': location_id,
                  'name' : 'cpu_sensor_01',
                  'unit' : 'percent'
                }
```

Executar o sql query

```
In [14]: cursor.execute(sql, data)
         sensor_id = cursor.lastrowid
         cnx.commit()
```

Readings E agora, obter alguns dados e enviar para a base de dados. Neste caso vamos usar a biblioteca `psutil` que permite obter informações sobre processos em execução e utilização do sistema (CPU, memória, discos, rede, sensores) em Python. (https://pypi.org/project/psutil/)

```
In [15]: import psutil

         sql = '''
         INSERT INTO Reading
             (idReading, idSensor, timestamp, value)
         VALUES
             (DEFAULT, %(idSensor)s, DEFAULT, %(value)s)
         '''

         for _ in range(20):
             data = {
                    'idSensor' : sensor_id,
                     'value' : psutil.cpu_percent(interval=1)
                    }
             cursor.execute(sql, data)
             cnx.commit()
             print('.', end='')

...

In [16]: cursor.close()
         cnx.close()
```

1.4 Selecionar dados em MySQL

Todo o processo é simples dados os conhecimentos anteriores

```
In [17]: cnx = mysql.connector.connect(**config)
         cursor = cnx.cursor()

In [18]: sql = '''
             SELECT idLocation, name, description
             FROM Location
             WHERE description LIKE "%163%"
         '''

         cursor.execute(sql)
```

E percorrer os dados, usando por exemplo um ciclo `for`

```
In [19]: for (idLocation, name, description) in cursor:
           print("id: {}\n\t name: {} \n\t description: {}".format(idLocation, name, descriptio

id: 1
         name: Prometheus Server
         description: Prometheus Server @ lab. 163 / ISE /UAlg
```

Outro exemplo

```
In [20]: sql = '''
             SELECT idReading, idSensor, timestamp, value
             FROM Reading
             WHERE value BETWEEN %s and %s
         '''
         data = (5, 50)

         cursor.execute(sql, data)

         for (idReading, idSensor, timestamp, value) in cursor:
           print("idReading: {}\n\t idSensor: {} \n\t time: {} \n\t value: {}".format(idReading

idReading: 1
         idSensor: 1
         time: 2020-03-13 15:25:15
         value: 24.4
idReading: 2
         idSensor: 1
         time: 2020-03-13 15:25:16
         value: 19.5
idReading: 3
         idSensor: 1
         time: 2020-03-13 15:25:17
         value: 20.4
idReading: 4
         idSensor: 1
         time: 2020-03-13 15:25:18
         value: 22.9
idReading: 5
         idSensor: 1
         time: 2020-03-13 15:25:19
         value: 11.7
idReading: 6
         idSensor: 1
         time: 2020-03-13 15:25:20
         value: 8.8
idReading: 7
         idSensor: 1
         time: 2020-03-13 15:25:21
         value: 14.3
idReading: 8
         idSensor: 1
         time: 2020-03-13 15:25:22
         value: 12.1
idReading: 9
         idSensor: 1
         time: 2020-03-13 15:25:23
         value: 10.2
```

```
idReading: 10
         idSensor: 1
         time: 2020-03-13 15:25:24
         value: 15.3
idReading: 11
         idSensor: 1
         time: 2020-03-13 15:25:25
         value: 11.6
idReading: 12
         idSensor: 1
         time: 2020-03-13 15:25:26
         value: 11.2
idReading: 13
         idSensor: 1
         time: 2020-03-13 15:25:27
         value: 9.0
idReading: 14
         idSensor: 1
         time: 2020-03-13 15:25:28
         value: 8.5
idReading: 15
         idSensor: 1
         time: 2020-03-13 15:25:29
         value: 9.5
idReading: 16
         idSensor: 1
         time: 2020-03-13 15:25:30
         value: 9.3
idReading: 17
         idSensor: 1
         time: 2020-03-13 15:25:31
         value: 9.9
idReading: 18
         idSensor: 1
         time: 2020-03-13 15:25:32
         value: 12.2
idReading: 19
         idSensor: 1
         time: 2020-03-13 15:25:33
         value: 10.3
idReading: 20
         idSensor: 1
         time: 2020-03-13 15:25:34
         value: 10.2
```

```
In [21]: sql = '''
             select *
```

```
            from Location
                inner join Sensor S on Location.idLocation = S.idLocation
                inner join Unit U on S.unit = U.unit
                inner join Reading R on S.idSensor = R.idSensor
            where value between %(low)s and %(high)s
            order by value
        '''

        data = {
            'low': 5,
            'high': 20
        }

        cursor.execute(sql, data)
```

Podemos obter os nomes e outros dados das colunas

```
In [22]: from mysql.connector import FieldType

         for i in range(len(cursor.description)):
             print("Column {}:".format(i+1))
             desc = cursor.description[i]
             print("  column_name = {}".format(desc[0]))
             print("  type = {} ({})".format(desc[1], FieldType.get_info(desc[1])))
             print("  null_ok = {}".format(desc[6]))
```

```
Column 1:
  column_name = idLocation
  type = 3 (LONG)
  null_ok = 0
Column 2:
  column_name = name
  type = 253 (VAR_STRING)
  null_ok = 0
Column 3:
  column_name = description
  type = 253 (VAR_STRING)
  null_ok = 0
Column 4:
  column_name = idSensor
  type = 3 (LONG)
  null_ok = 0
Column 5:
  column_name = idLocation
  type = 3 (LONG)
  null_ok = 0
Column 6:
  column_name = name
```

```
  type = 253 (VAR_STRING)
  null_ok = 0
Column 7:
  column_name = unit
  type = 253 (VAR_STRING)
  null_ok = 0
Column 8:
  column_name = unit
  type = 253 (VAR_STRING)
  null_ok = 0
Column 9:
  column_name = description
  type = 253 (VAR_STRING)
  null_ok = 0
Column 10:
  column_name = idReading
  type = 3 (LONG)
  null_ok = 0
Column 11:
  column_name = idSensor
  type = 3 (LONG)
  null_ok = 0
Column 12:
  column_name = timestamp
  type = 7 (TIMESTAMP)
  null_ok = 0
Column 13:
  column_name = value
  type = 4 (FLOAT)
  null_ok = 0
```

```
In [23]: lista_de_colunas = [linha[0] for linha in cursor.description]
         lista_de_colunas

Out[23]: ['idLocation',
          'name',
          'description',
          'idSensor',
          'idLocation',
          'name',
          'unit',
          'unit',
          'description',
          'idReading',
          'idSensor',
          'timestamp',
          'value']
```

```
In [24]: for linha in cursor:
             print('\t'.join([f'|{coluna}: {valor}' for valor, coluna  in zip(linha, lista_de_c
```

```
|idLocation: 1        |name: Prometheus Server        |description: Prometheus Server @ lab. 16
|idLocation: 1        |name: Prometheus Server        |description: Prometheus Server @ lab. 16
|idLocation: 1        |name: Prometheus Server        |description: Prometheus Server @ lab. 16
|idLocation: 1        |name: Prometheus Server        |description: Prometheus Server @ lab. 16
|idLocation: 1        |name: Prometheus Server        |description: Prometheus Server @ lab. 16
|idLocation: 1        |name: Prometheus Server        |description: Prometheus Server @ lab. 16
|idLocation: 1        |name: Prometheus Server        |description: Prometheus Server @ lab. 16
|idLocation: 1        |name: Prometheus Server        |description: Prometheus Server @ lab. 16
|idLocation: 1        |name: Prometheus Server        |description: Prometheus Server @ lab. 16
|idLocation: 1        |name: Prometheus Server        |description: Prometheus Server @ lab. 16
|idLocation: 1        |name: Prometheus Server        |description: Prometheus Server @ lab. 16
|idLocation: 1        |name: Prometheus Server        |description: Prometheus Server @ lab. 16
|idLocation: 1        |name: Prometheus Server        |description: Prometheus Server @ lab. 16
|idLocation: 1        |name: Prometheus Server        |description: Prometheus Server @ lab. 16
|idLocation: 1        |name: Prometheus Server        |description: Prometheus Server @ lab. 16
|idLocation: 1        |name: Prometheus Server        |description: Prometheus Server @ lab. 16
|idLocation: 1        |name: Prometheus Server        |description: Prometheus Server @ lab. 16
```

1.4.1 o comando `fetchall`

Usando o comando `fetchall` podemos obter todos os resultados de uma única vez como uma lista de tuplos

```
In [25]: # é necessario voltar a correr o select pois o cursor foi esvaziado
         cursor.execute(sql, data)

         cursor.fetchall()
```

```
Out[25]: [(1,
           'Prometheus Server',
           'Prometheus Server @ lab. 163 / ISE /UAlg',
           1,
           1,
           'cpu_sensor_01',
           'percent',
           'percent',
           'percentage of usage',
           14,
           1,
           datetime.datetime(2020, 3, 13, 15, 25, 28),
           8.5),
          (1,
           'Prometheus Server',
           'Prometheus Server @ lab. 163 / ISE /UAlg',
```

```
 1,
 1,
 'cpu_sensor_01',
 'percent',
 'percent',
 'percentage of usage',
 6,
 1,
 datetime.datetime(2020, 3, 13, 15, 25, 20),
 8.8),
(1,
 'Prometheus Server',
 'Prometheus Server @ lab. 163 / ISE /UAlg',
 1,
 1,
 'cpu_sensor_01',
 'percent',
 'percent',
 'percentage of usage',
 13,
 1,
 datetime.datetime(2020, 3, 13, 15, 25, 27),
 9.0),
(1,
 'Prometheus Server',
 'Prometheus Server @ lab. 163 / ISE /UAlg',
 1,
 1,
 'cpu_sensor_01',
 'percent',
 'percent',
 'percentage of usage',
 16,
 1,
 datetime.datetime(2020, 3, 13, 15, 25, 30),
 9.3),
(1,
 'Prometheus Server',
 'Prometheus Server @ lab. 163 / ISE /UAlg',
 1,
 1,
 'cpu_sensor_01',
 'percent',
 'percent',
 'percentage of usage',
 15,
 1,
 datetime.datetime(2020, 3, 13, 15, 25, 29),
```

```
 9.5),
(1,
 'Prometheus Server',
 'Prometheus Server @ lab. 163 / ISE /UAlg',
 1,
 1,
 'cpu_sensor_01',
 'percent',
 'percent',
 'percentage of usage',
 17,
 1,
 datetime.datetime(2020, 3, 13, 15, 25, 31),
 9.9),
(1,
 'Prometheus Server',
 'Prometheus Server @ lab. 163 / ISE /UAlg',
 1,
 1,
 'cpu_sensor_01',
 'percent',
 'percent',
 'percentage of usage',
 20,
 1,
 datetime.datetime(2020, 3, 13, 15, 25, 34),
 10.2),
(1,
 'Prometheus Server',
 'Prometheus Server @ lab. 163 / ISE /UAlg',
 1,
 1,
 'cpu_sensor_01',
 'percent',
 'percent',
 'percentage of usage',
 9,
 1,
 datetime.datetime(2020, 3, 13, 15, 25, 23),
 10.2),
(1,
 'Prometheus Server',
 'Prometheus Server @ lab. 163 / ISE /UAlg',
 1,
 1,
 'cpu_sensor_01',
 'percent',
 'percent',
```

```
 'percentage of usage',
 19,
 1,
 datetime.datetime(2020, 3, 13, 15, 25, 33),
 10.3),
(1,
 'Prometheus Server',
 'Prometheus Server @ lab. 163 / ISE /UAlg',
 1,
 1,
 'cpu_sensor_01',
 'percent',
 'percent',
 'percentage of usage',
 12,
 1,
 datetime.datetime(2020, 3, 13, 15, 25, 26),
 11.2),
(1,
 'Prometheus Server',
 'Prometheus Server @ lab. 163 / ISE /UAlg',
 1,
 1,
 'cpu_sensor_01',
 'percent',
 'percent',
 'percentage of usage',
 11,
 1,
 datetime.datetime(2020, 3, 13, 15, 25, 25),
 11.6),
(1,
 'Prometheus Server',
 'Prometheus Server @ lab. 163 / ISE /UAlg',
 1,
 1,
 'cpu_sensor_01',
 'percent',
 'percent',
 'percentage of usage',
 5,
 1,
 datetime.datetime(2020, 3, 13, 15, 25, 19),
 11.7),
(1,
 'Prometheus Server',
 'Prometheus Server @ lab. 163 / ISE /UAlg',
 1,
```

```
 1,
 'cpu_sensor_01',
 'percent',
 'percent',
 'percentage of usage',
 8,
 1,
 datetime.datetime(2020, 3, 13, 15, 25, 22),
 12.1),
(1,
 'Prometheus Server',
 'Prometheus Server @ lab. 163 / ISE /UAlg',
 1,
 1,
 'cpu_sensor_01',
 'percent',
 'percent',
 'percentage of usage',
 18,
 1,
 datetime.datetime(2020, 3, 13, 15, 25, 32),
 12.2),
(1,
 'Prometheus Server',
 'Prometheus Server @ lab. 163 / ISE /UAlg',
 1,
 1,
 'cpu_sensor_01',
 'percent',
 'percent',
 'percentage of usage',
 7,
 1,
 datetime.datetime(2020, 3, 13, 15, 25, 21),
 14.3),
(1,
 'Prometheus Server',
 'Prometheus Server @ lab. 163 / ISE /UAlg',
 1,
 1,
 'cpu_sensor_01',
 'percent',
 'percent',
 'percentage of usage',
 10,
 1,
 datetime.datetime(2020, 3, 13, 15, 25, 24),
 15.3),
```

```
(1,
 'Prometheus Server',
 'Prometheus Server @ lab. 163 / ISE /UAlg',
 1,
 1,
 'cpu_sensor_01',
 'percent',
 'percent',
 'percentage of usage',
 2,
 1,
 datetime.datetime(2020, 3, 13, 15, 25, 16),
 19.5)]
```

Podemos também converter para um dicionário mas **nosso caso NÃO é boa ideia** pois há colunas que "têm o mesmo nome" (e.g., `nome`), pelo que se perdem colunas ao passar para um dicionário.

```
In [26]: # é necessario voltar a correr o select pois o cursor foi esvaziado
         cursor.execute(sql, data)

         for linha in cursor:
             print({coluna: valor for valor, coluna  in zip(linha, lista_de_colunas)})
```

```
{'idLocation': 1, 'name': 'cpu_sensor_01', 'description': 'percentage of usage', 'idSensor': 1
{'idLocation': 1, 'name': 'cpu_sensor_01', 'description': 'percentage of usage', 'idSensor': 1
{'idLocation': 1, 'name': 'cpu_sensor_01', 'description': 'percentage of usage', 'idSensor': 1
{'idLocation': 1, 'name': 'cpu_sensor_01', 'description': 'percentage of usage', 'idSensor': 1
{'idLocation': 1, 'name': 'cpu_sensor_01', 'description': 'percentage of usage', 'idSensor': 1
{'idLocation': 1, 'name': 'cpu_sensor_01', 'description': 'percentage of usage', 'idSensor': 1
{'idLocation': 1, 'name': 'cpu_sensor_01', 'description': 'percentage of usage', 'idSensor': 1
{'idLocation': 1, 'name': 'cpu_sensor_01', 'description': 'percentage of usage', 'idSensor': 1
{'idLocation': 1, 'name': 'cpu_sensor_01', 'description': 'percentage of usage', 'idSensor': 1
{'idLocation': 1, 'name': 'cpu_sensor_01', 'description': 'percentage of usage', 'idSensor': 1
{'idLocation': 1, 'name': 'cpu_sensor_01', 'description': 'percentage of usage', 'idSensor': 1
{'idLocation': 1, 'name': 'cpu_sensor_01', 'description': 'percentage of usage', 'idSensor': 1
{'idLocation': 1, 'name': 'cpu_sensor_01', 'description': 'percentage of usage', 'idSensor': 1
{'idLocation': 1, 'name': 'cpu_sensor_01', 'description': 'percentage of usage', 'idSensor': 1
{'idLocation': 1, 'name': 'cpu_sensor_01', 'description': 'percentage of usage', 'idSensor': 1
{'idLocation': 1, 'name': 'cpu_sensor_01', 'description': 'percentage of usage', 'idSensor': 1
{'idLocation': 1, 'name': 'cpu_sensor_01', 'description': 'percentage of usage', 'idSensor': 1
```

```
In [27]: cursor.close()
         cnx.close()
```

1.4.2 Dados na forma de dicionários

Se criar o cursos com o parametro `dictionary=True` ao iterar sobre os resultados estes vêm na forma de dicionários

```
In [28]: cnx = mysql.connector.connect(**config)
         cursor = cnx.cursor(dictionary=True)

         sql = '''
             SELECT idReading, idSensor, timestamp, value
             FROM Reading
             WHERE value BETWEEN %s and %s
         '''
         data = (5, 50)

         cursor.execute(sql, data)

         for linha in cursor:
             print(linha)
```

```
{'idReading': 1, 'idSensor': 1, 'timestamp': datetime.datetime(2020, 3, 13, 15, 25, 15), 'value
{'idReading': 2, 'idSensor': 1, 'timestamp': datetime.datetime(2020, 3, 13, 15, 25, 16), 'value
{'idReading': 3, 'idSensor': 1, 'timestamp': datetime.datetime(2020, 3, 13, 15, 25, 17), 'value
{'idReading': 4, 'idSensor': 1, 'timestamp': datetime.datetime(2020, 3, 13, 15, 25, 18), 'value
{'idReading': 5, 'idSensor': 1, 'timestamp': datetime.datetime(2020, 3, 13, 15, 25, 19), 'value
{'idReading': 6, 'idSensor': 1, 'timestamp': datetime.datetime(2020, 3, 13, 15, 25, 20), 'value
{'idReading': 7, 'idSensor': 1, 'timestamp': datetime.datetime(2020, 3, 13, 15, 25, 21), 'value
{'idReading': 8, 'idSensor': 1, 'timestamp': datetime.datetime(2020, 3, 13, 15, 25, 22), 'value
{'idReading': 9, 'idSensor': 1, 'timestamp': datetime.datetime(2020, 3, 13, 15, 25, 23), 'value
{'idReading': 10, 'idSensor': 1, 'timestamp': datetime.datetime(2020, 3, 13, 15, 25, 24), 'valu
{'idReading': 11, 'idSensor': 1, 'timestamp': datetime.datetime(2020, 3, 13, 15, 25, 25), 'valu
{'idReading': 12, 'idSensor': 1, 'timestamp': datetime.datetime(2020, 3, 13, 15, 25, 26), 'valu
{'idReading': 13, 'idSensor': 1, 'timestamp': datetime.datetime(2020, 3, 13, 15, 25, 27), 'valu
{'idReading': 14, 'idSensor': 1, 'timestamp': datetime.datetime(2020, 3, 13, 15, 25, 28), 'valu
{'idReading': 15, 'idSensor': 1, 'timestamp': datetime.datetime(2020, 3, 13, 15, 25, 29), 'valu
{'idReading': 16, 'idSensor': 1, 'timestamp': datetime.datetime(2020, 3, 13, 15, 25, 30), 'valu
{'idReading': 17, 'idSensor': 1, 'timestamp': datetime.datetime(2020, 3, 13, 15, 25, 31), 'valu
{'idReading': 18, 'idSensor': 1, 'timestamp': datetime.datetime(2020, 3, 13, 15, 25, 32), 'valu
{'idReading': 19, 'idSensor': 1, 'timestamp': datetime.datetime(2020, 3, 13, 15, 25, 33), 'valu
{'idReading': 20, 'idSensor': 1, 'timestamp': datetime.datetime(2020, 3, 13, 15, 25, 34), 'valu
```

```
In [29]: cursor.close()
         cnx.close()
```