02_MySQL

March 16, 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 0x7f6f8141a4e0>
```

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

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

```
Out[2]: {'_cmysql': <_mysql_connector.MySQL at 0x2c89430>,
         '_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': 141,
          '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
        import mysql.connector

        from config import config
        print(config)

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

```
{'host': 'localhost', 'user': 'sensors', 'password': '##sensors##', 'db': 'sensors'}
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 (reinicie o kernel) e corra a linha acima: qual a mensagem de erro?
3. Re-implemente o código de modo a quando falar pelo servidor estar desligado, dar a mensagem adequada e voltar a tentar mais 2 vezes a cada 5 segundos (vejam o pacote `time` e particular o método `sleep()`)

```
In [5]: import time
        print('ola')
        time.sleep(5)
        print('ola de novo!')
```

```
ola
ola de novo!
```

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 [6]:
```
# from config import config as conf
from config import config
import mysql.connector

sql = '''
    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
```

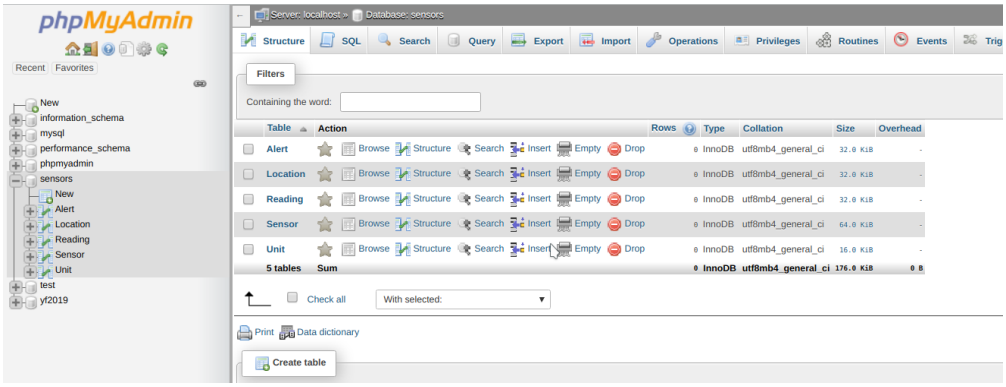
```
    ON UPDATE CASCADE) ENGINE = InnoDB;


-- -----------------------------------------------------
-- 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 [7]: 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:
```



phpmyadmin_database_created.png

```
        print(err)
    else:
        print('ok!')
        cnx.close()
```

```
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 [8]: 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" - ver https://pyformat.info/) e um tuplo com os dados

```
In [9]: 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')
```

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

```
In [10]: cursor.execute(sql, data)
         location_id = cursor.lastrowid
         location_id


         ---------------------------------------------------------------------------

         MySQLInterfaceError                       Traceback (most recent call last)

         /home/pcardoso/.local/lib/python3.7/site-packages/mysql/connector/connection_cext.py in
         488                                 raw=raw, buffered=buffered,
     --> 489                                 raw_as_string=raw_as_string)
         490         except MySQLInterfaceError as exc:


         MySQLInterfaceError: Duplicate entry 'Prometheus Server' for key 'name_UNIQUE'


     During handling of the above exception, another exception occurred:


         IntegrityError                            Traceback (most recent call last)

         <ipython-input-10-32e6a17d62f7> in <module>()
     ----> 1 cursor.execute(sql, data)
           2 location_id = cursor.lastrowid
           3 location_id


         /home/pcardoso/.local/lib/python3.7/site-packages/mysql/connector/cursor_cext.py in exe
         264             result = self._cnx.cmd_query(stmt, raw=self._raw,
         265                                          buffered=self._buffered,
     --> 266                                          raw_as_string=self._raw_as_string)
         267         except MySQLInterfaceError as exc:
         268             raise errors.get_mysql_exception(msg=exc.msg, errno=exc.errno,


         /home/pcardoso/.local/lib/python3.7/site-packages/mysql/connector/connection_cext.py in
         490         except MySQLInterfaceError as exc:
         491             raise errors.get_mysql_exception(exc.errno, msg=exc.msg,
     --> 492                                              sqlstate=exc.sqlstate)
         493         except AttributeError:
         494             if self._unix_socket:
```

```
IntegrityError: 1062 (23000): Duplicate entry 'Prometheus Server' for key 'name_UNIQUE
```

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 [ ]: cnx.commit()
```

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

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

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

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

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

Executar o sql query

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

In [ ]: sensor_id
```

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

1.4 Selecionar dados em MySQL

Todo o processo é simples dados os conhecimentos anteriores

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

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

Outros exemplos

```
In [ ]: 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(idReadi
```

```
In [ ]: 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': 40
        }

        cursor.execute(sql, data)
```

Podemos obter os nomes e outros dados das colunas (https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlcursor-description.html) no formato (name, type_code, display_size, internal_size, precision, scale, null_ok, column_flags)

```
In [ ]: cursor.description
```

```
In [ ]: 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]))
```

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

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

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

        cursor.fetchall()
```

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 [ ]: # 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)})
```

```
In [ ]: 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 [ ]: 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)
```

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

1.4.3 EXERCÍCIO

Utilize os pacotes `time` e `psutil` para calcular a memoria virtual livre a cada segundo durante 30 segundos, guardando na base de dados. Use

```
...
x = psutil.virtual_memory()
x.free
...
```