khadga19024_A2_DBMS

March 20, 2022

[1]:
```python
import mysql.connector
from mysql.connector import errorcode


class Connections:
    """You can either supply the configuration dictionary for MySQL access or␣
 ↪you can give the details
    To get a config file use function 'get_config'
    config = {
                "user": user_name (e.g "root"),
                "password": your password,
                "host": "localhost" ,
                "database": None,
                "raise_on_warnings": True}"""

    def __init__(self, config: dict = None):
        if config is not None:
            self.config = config
        else:
            user = str(input("use default (root@localhost)(Y/N): "))
            if user.lower() in ["y", "yes"]:
                self.config = {
                    "user": "root",
                    "password": str(input("root password: ")),
                    "host": "localhost",
                    "database": None,
                    "raise_on_warnings": True,
                }
            else:
                self.config = {
                    "user": str(input("user: ")),
                    "password": str(input("user password: ")),
                    "host": "localhost"
                    if (host := str(input('host ("l" for localhost): '))) == "l"
                    else host,
                    "database": None,
                    "raise_on_warnings": True,
                }
        self.cnx = mysql.connector.connect(**self.config)

        print("OK")

        self.cursor = self.cnx.cursor(buffered=True)

    def get_config(self):
        """Returns a dictionary config file of the current configuration
        example:
            config = {
                    "user": "root",
                    "password": 123,
                    "host": "localhost" ,
                    "database": None,
                    "raise_on_warnings": True}"""
        return self.config

    def GODMODE(self, passthrough, database=None):
        """Pass any query which can be passed to mysql console"""
        if database is not None:
            self.cnx.database = database
        try:
            self.cursor.execute(passthrough)
            print("executed")
        except mysql.connector.Error as err:
            print(err)
        self.cnx.commit()

        try:
            return self.cursor.fetchall()
        except Exception as e:
            pass
#          self.cnx.close()

    def create_db(self, database):
        """Function to create a database"""
        query = f"Create database {database};"
        print(f"{query}")
        try:
            self.cursor.execute(query)
            print("executed")
            self.cursor.execute(f"use {database};")
        except mysql.connector.Error as err:
            print(err)
            if err.errno == errorcode.ER_DB_CREATE_EXISTS:
                print("Already exists")
```

```python
        self.cnx.database = database

    def CREATE(self, database=None, table=None, columns=None, passthrough=None):
        """General Purpose CREATE operation
        example:
            CREATE(database = "Classes",table = "Courses",columns =␣
↪{'course_name':'varchar(25)','Course_id':'varchar(7)'})

        """
        if database is not None:
            self.create_db(database)
        if table is not None:
            if self.cnx.database == None:
                print("No database Selected")
                database = str(input("select or create database :"))
                create_db(self, database)
            self.cursor.execute(f"show tables;")
            c = self.cursor.fetchall()
            t = []
            for x in c:
                #                 print(x)
                if x[0] == table.lower():
                    print("table already exists")
                    return None

            if columns == None:
                columns = {}
                print("TO create a table atleast 1 column is necessary")
                i = 1
                while True:
                    key = str(input(f"Column {i} Name :"))
                    if key == "":
                        break
                    value = str(input(f"Column {i} constraint :"))
                    while value == "":
                        print("Constraint is necessary")
                        value = str(input(f"Column {i} constraint :"))

                    columns[key] = value
                    i += 1
            if columns is not None:
                col = ""
                for key in columns.keys():
                    col = col + f"{key} {columns[key]},"
                col = col[:-1]
            query = f"Create table {table}({col});"
            print(f"{query}")
            try:
                self.cursor.execute(query)
                print("executed")
            except mysql.connector.Error as err:
                print(err)
                if err.errno == errorcode.ER_TABLE_EXISTS_ERROR:
                    print("Already exists")
#          self.cnx.close()

    def drop_table(self, database, table):
        """DROP a particular table from a database"""
        query = f"DROP table {table};"
        print(f"{query}")
        try:
            self.cnx.database = database
            self.cursor.execute(query)
            print("executed")

        except mysql.connector.Error as err:
            print(err)
            if err.errno == errorcode.ER_TABLE_NAME:
                print("Database doesnt exists")

    def DROP(self, database, table=None, passthrough=None):
        """General purpose DROP Operation"""
        if passthrough is not None:
            self.GODMODE(passthrough, database=database)
        else:
            if table is not None:

                self.drop_table(database=database, table=table)
                return None
            query = f"DROP database {database};"
            print(f"{query}")
            try:
                self.cursor.execute(query)
                print("executed")

            except mysql.connector.Error as err:
                print(err)
                if err.errno == errorcode.ER_DB_DROP_EXISTS:
                    print("Database doesnt exists")

#          self.cnx.close()

    def ALTER_TABLE(
        self, database, table=None, columns=None, modifier=None,␣
↪passthrough=None
    ):
        """GENERAL Purpose ALTER TABLE Operation
        example:
            ALTER_TABLE('Classes',table='Courses',columns={'course_name':
↪'varchar(50)'},modifier='modify')"""
        if passthrough is not None:
            self.GODMODE(passthrough, database=database)
        else:
            query = f"ALTER TABLE {table}"
            self.cnx.database = database
            if type(columns) == dict:
                col = ""
                for key in columns.keys():
                    col = col + f" {modifier} {key} {columns[key]},"
                col = col[:-1]

            elif type(columns) == list or type(columns) == tuple:
                col = ""
                for key in columns:
                    col = col + f" {modifier} {key},"
                col = col[:-1]
            else:
                col = f" {modifier} {columns};"
            query = query + col
            print(f"{query}")
            try:
                self.cursor.execute(query)
                print("executed")

            except mysql.connector.Error as err:
                print(err)

#              self.cnx.close()

    def SELECT(self, database, query=None):
        """General Purpose SELECT Operation
        example:
            SELECT(database  = 'Classes', query = 'Select * from courses where␣
↪course_name = "DBMS"')"""
#          self.cnx.reconnect()
        self.cnx.database = database
        print(f"{query}")
        try:
            self.cursor.execute(query)
            print("executed")
            col = self.cursor.column_names
            print(self.cursor.column_names)
            for col in self.cursor:
                print(col)

        except mysql.connector.Error as err:
            print(err)
#          self.cnx.close()

    def INSERT(self, database, table=None, values=None, passthrough=None):
        """General Purpose INSERT Operation
        example:
            INSERT(database = "Classes",table = 'Courses',values =␣
↪(("DBMS,'DSE-312'"),('DSML','DSE-302'),('ALGORITHMS','DSE-304')))

            INSERT(database = "Classes",table = 'Courses',values =␣
↪("DBMS",'DSE-312'))"""
        if passthrough is not None:
            self.GODMODE(passthrough, database=database)
        else:
            self.cnx.database = database

            self.cursor.execute(f"select * from {table} limit 1")
            col_names = self.cursor.column_names

            col_name = ""
            for cname in col_names:
                col_name += f" {cname},"
#              print(col_name[0:-1])
            query = f"INSERT INTO {table} ({col_name[0:-1]}) VALUES "
            if any(isinstance(i, list) or isinstance(i, tuple) for i in values):
                if len(values[0]) == 1:
                    for val in values:
                        query = query + f"({val[0]}),"
                else:
                    for val in values:
                        query = query + f"{val},"

            else:
                if len(values) == 1:
                    for val in values:
                        query = query + f"({val}),"
                else:
                    query = query + f"{values},"

            query = query[0:-1]
            print(query)
            try:
                self.cursor.execute(query)
                print("executed")

            except mysql.connector.Error as err:
                print(err)

        self.cnx.commit()
#         self.cnx.close()

    def UPDATE(self, database, table=None, columns=None, where=None,␣
↪passthrough=None):
        """UPDATE Operation
        example:
            UPDATE(database = 'Classes',table = 'Courses',columns =␣
↪{'course_id':'DSE-310'},where = 'where course_name = "DBMS"')"""
        self.cnx.database = database
        if passthrough is not None:
            self.GODMODE(passthrough, database=database)
        else:
            query = f"UPDATE {table} set "
            col = ""
            for key in columns.keys():
                col = col + f' {key} = "{columns[key]}",'
            col = col[0:-1]
            col = col + " " + where
            query = query + col + ";"
            print(query)
            try:
                self.cursor.execute(query)
                print("executed")
            except mysql.connector.Error as err:
                print(err)
        self.cnx.commit()
#         self.cnx.close()

    def NOTNULL(self, database, table=None, columns=None, passthrough=None):
        """Make a column NOT NULL
        example:
            NOTNULL(database = "Classes",table = "Courses",columns =␣
↪{'course_name':'varchar(30)','course_id':'varchar(7)'})"""
        if passthrough is not None:
            self.GODMODE(passthrough, database=database)
        else:
            for key in columns.keys():
                columns[key] = f"{columns[key]} NOT NULL"
            #             print(columns)a
            self.ALTER_TABLE(
                database, table, columns, modifier="modify", passthrough=None
            )
        self.cnx.commit()
#         self.cnx.close()

    def PRIMARY_KEY(
        self, database, table=None, columns=None, constraint_name=None,␣
↪passthrough=None
    ):
        """Define a PRIMARY KEY
        example:
            PRIMARY_KEY(database = 'Classes',table = 'Courses',columns =␣
↪('course_id'))"""
        if passthrough is not None:
            self.GODMODE(passthrough, database=database)
        else:

            print(columns)
            query = f"ALTER TABLE {table} ADD "
            if constraint_name is not None:
                query += f"CONSTRAINT {constraint_name} "
            query += "PRIMARY KEY ("
            if type(columns) == str:
                query = query + f"{columns})"

            else:
                if len(columns) == 1:
                    query = query + f"{columns[0]}, "
                #                     print('here',query)
                else:
                    for col in columns:
                        query = query + f"{col}, "
                query = query[0:-2] + ")"
            print(query)
            self.ALTER_TABLE(database, passthrough=query)

    def FOREIGN_KEY(
        self, database, table=None, columns=None, constraint_name=None,␣
↪passthrough=None
    ):
        """Define a FOREIGN KEY
        example:
            CREATE(database = "Classes",table='instructor', columns={'inst_id' :
↪'int not null','course_id' :'varchar(7)'})
            PRIMARY_KEY(database = 'Classes',table = 'instructor',columns =␣
↪('inst_id'))
```

```
            FOREIGN_KEY(database = "Classes",table = "instructor", columns =␣
↪{"Courses": "course_id"}, constraint_name = {'Courses':"cid_fkey"})
        """
        if passthrough is not None:
            self.GODMODE(passthrough, database=database)
        else:
            print(columns)
            query = f"ALTER TABLE {table}"

            #             query+='FOREIGN KEY ('
            for key in columns.keys():
                query += " ADD"
                if constraint_name[key] is not None:
                    query += f" CONSTRAINT {constraint_name[key]}"

                query += " FOREIGN KEY ("
                q = ""
                if type(columns[key]) is not str:
                    for val in columns[key]:
                        q = q + f"{val}, "
                    q = q[0:-2]
                else:
                    q = q + f"{columns[key]}"
                print(q)
                query = query + q + ") REFERENCES "

                query = query + f"{key} ({q}), "
            query = query[0:-2]
            print(query)
            self.ALTER_TABLE(database, passthrough=query)

    def new_user(self, user_name, user_pwd, ip="localhost"):
        """example:
            new_user(user_name='test_user1',user_pwd='test_user_pwd')"""
        query = f'CREATE USER {user_name}@{ip} IDENTIFIED BY "{user_pwd}";'
        print(query)
        self.GODMODE(passthrough=query)

    def drop_user(self, user_name, ip="localhost"):
        query = f"DROP USER"
        if type(user_name) is dict:
            for name, ip in zip(user_name.keys(), user_name.values()):
                query += f" {name}@{ip}, "
            query = query[0:-2]
        elif type(user_name) is not str:
            for name in user_name:
                query += f" {name}@{ip}, "
            query = query[0:-2]
        else:
            query += f" {user_name}@{ip}"
        print(query)
        self.GODMODE(passthrough=query)

    def grant_priv(
        self, user_name, privlage=None, database=None, table=None,␣
↪ip="localhost"
    ):
        """GRANT privlages to user@ip
        privalge : ['ALL','SELECT','INSERT','UPDATE', ...]
                    to see full list goto. (https://dev.mysql.com/doc/refman/8.
↪0/en/privileges-provided.html)
        example:
           ␣
↪grant_priv(user_name='test_user1',privlage='ALL',database='Classes')"""
        priv = ", ".join(privlage) if type(privlage) is not str else privlage
        database = "*" if database is None else database
        table = "*" if table is None else table
        query = f"GRANT {priv} ON {database}.{table} TO {user_name}@{ip}"
        print(query)
        self.GODMODE(passthrough=query)

    def revoke_priv(
        self, user_name, revoke=None, database=None, table=None, ip="localhost"
    ):
        """REVOKE privlage from user@ip
        see (https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html)␣
↪to find all privlages
        example:
            revoke_priv(user_name='test_user1',revoke='SELECT')"""
        rev = ", ".join(revoke) if type(revoke) is not str else revoke
        database = "*" if database is None else database
        table = "*" if table is None else table
        query = f"REVOKE {rev} ON {database}.{table} FROM {user_name}@{ip}"
        print(query)
        self.GODMODE(passthrough=query)
    def reconnect(self):
        self.__init__()

# Connections().connect().USERS().new_user('akj1','AKJ')

# if __name__=='__main__':
#     Connections()
```

0.0.1 SOME EXAMPLES:

[2]:
```
cfg = {
    "user": "root",
    "password": "", # Your password
    "host": "localhost",
    "database": None,
    "raise_on_warnings": True,
}
test_usr_config = {
    "user": "test_user1",
    "password": "test_user_pwd",
    "host": "localhost",
    "database": None,
    "raise_on_warnings": True,
}
```

[3]:
```
con = Connections(cfg)
```

```
OK
```

[4]:
```
con.CREATE(database = "Classes",table = "Courses",columns = {'course_name':
 ↪'varchar(25)','Course_id':'varchar(7)'})
```

```
Create database Classes;
executed
Create table Courses(course_name varchar(25),Course_id varchar(7));
executed
```

[5]:
```
con.INSERT(database = "Classes",table = 'Courses',values =␣
 ↪(('DSML','DSE-302'),('ALGORITHMS','DSE-304')))
```

```
INSERT INTO Courses ( course_name, Course_id) VALUES ('DSML',
'DSE-302'),('ALGORITHMS', 'DSE-304')
executed
```

[6]:
```
con.SELECT(database  = 'Classes', query = 'Select * from courses ')
```

```
Select * from courses
executed
('course_name', 'Course_id')
('DSML', 'DSE-302')
('ALGORITHMS', 'DSE-304')
```

[7]:
```
con.INSERT(database = "Classes",table = 'Courses',values = ("DBMS",'DSE-312'))
```

```
INSERT INTO Courses ( course_name, Course_id) VALUES ('DBMS', 'DSE-312')
executed
```

[8]:
```
con.SELECT(database  = 'Classes', query = 'Select * from courses ')
```

```
Select * from courses
executed
('course_name', 'Course_id')
('DSML', 'DSE-302')
('ALGORITHMS', 'DSE-304')
('DBMS', 'DSE-312')
```

[9]:
```
con.UPDATE(database = 'Classes',table = 'Courses',columns = {'course_id':
 ↪'DSE-310'},where = 'where course_name = "DBMS"')
```

```
UPDATE Courses set  course_id = "DSE-310" where course_name = "DBMS";
executed
```

[10]:
```
con.SELECT(database  = 'Classes', query = 'Select * from courses where␣
 ↪course_name = "DBMS"')
```

```
Select * from courses where course_name = "DBMS"
executed
('course_name', 'Course_id')
('DBMS', 'DSE-310')
```

[11]:
```
con.ALTER_TABLE('Classes',table='Courses',columns={'course_name':
 ↪'varchar(50)'},modifier='modify')
```

```
ALTER TABLE Courses modify course_name varchar(50)
executed
```

[12]:
```
con.NOTNULL(database = "Classes",table = "Courses",columns = {'course_name':
 ↪'varchar(50)','course_id':'varchar(7)'})
```

```
ALTER TABLE Courses modify course_name varchar(50) NOT NULL, modify course_id
varchar(7) NOT NULL
executed
```

[13]:
```
con.PRIMARY_KEY(database = 'Classes',table = 'Courses',columns = ('course_id'))
```

```
course_id
ALTER TABLE Courses ADD PRIMARY KEY (course_id)
executed
```

[14]:
```
con.CREATE(database = "Classes",table='instructor', columns={'inst_id' :'int␣
 ↪not null','course_id' :'varchar(7)'})
con.PRIMARY_KEY(database = 'Classes',table = 'instructor',columns = ('inst_id'))
con.FOREIGN_KEY(database = "Classes",table = "instructor", columns = {"Courses":
 ↪ "course_id"}, constraint_name = {'Courses':"cid_fkey"})
```

```
Create database Classes;
1007 (HY000): Can't create database 'classes'; database exists
Already exists
Create table instructor(inst_id int not null,course_id varchar(7));
executed
```

```
inst_id
ALTER TABLE instructor ADD PRIMARY KEY (inst_id)
executed
{'Courses': 'course_id'}
course_id
ALTER TABLE instructor ADD CONSTRAINT cid_fkey FOREIGN KEY (course_id)
REFERENCES Courses (course_id)
executed
```

```
[15]: con.new_user(user_name='test_user1',user_pwd='test_user_pwd')
```

```
CREATE USER test_user1@localhost IDENTIFIED BY "test_user_pwd";
executed
```

```
[16]: con.grant_priv(user_name='test_user1',privlage='ALL',database='Classes')
```

```
GRANT ALL ON Classes.* TO test_user1@localhost
executed
```

```
[ ]:
```

```
[17]: # cfg_test_usr = con2.get_config()
```

```
[18]: con.SELECT(database='Classes',query = 'SELECT * from courses')
```

```
SELECT * from courses
executed
('course_name', 'course_id')
('DSML', 'DSE-302')
('ALGORITHMS', 'DSE-304')
('DBMS', 'DSE-310')
```

```
[19]: con.revoke_priv(user_name='test_user1',revoke='ALL')
```

```
REVOKE ALL ON *.* FROM test_user1@localhost
executed
```

```
[20]: con2 = Connections(test_usr_config)
```

```
OK
```

```
[21]: con2.SELECT(database='Classes',query = 'SELECT * from courses')
```

```
---------------------------------------------------------------------------
MySQLInterfaceError                       Traceback (most recent call last)
Input In [21], in <cell line: 1>()
----> 1 con2.SELECT(database='Classes',query = 'SELECT * from courses')

Input In [1], in Connections.SELECT(self, database, query)
    211         """General Purpose SELECT Operation
    212         example:
    213             SELECT(database  = 'Classes', query = 'Select * from courses␣
 ↪where course_name = "DBMS"')"""
    214 #         self.cnx.reconnect()
--> 215         self.cnx.database = database
    216         print(f"{query}")
    217         try:

File ~\anaconda3\lib\site-packages\mysql\connector\connection_cext.py:170, in␣
 ↪CMySQLConnection.database(self, value)
    167 @database.setter
    168 def database(self, value):  # pylint: disable=W0221
    169     """Set the current database"""
--> 170     self._cmysql.select_db(value)

MySQLInterfaceError: Access denied for user 'test_user1'@'localhost' to database␣
 ↪'classes'
```

[22]:
```
con.drop_user(user_name='test_user1') #RUN THIS
```

```
DROP USER test_user1@localhost
executed
```

[23]:
```
con.DROP(database = 'classes') #RUN THIS
```

```
DROP database classes;
executed
```

[ ]:

[ ]: