Demonstration of programmatic database access

This simple notebook shows how to establish a connection with a MySQL database created located somewhere on the Internet.

- It is partly based on the code from this tutorial https://dev.mysql.com/doc/connector-python/en/connector-python-examples.html
- It also uses the example ap database we used in class
- I am running it from Google Colaboratory https://colab.research.google.com/notebooks/welcome.ipynb. Some changes might be required if you are running it in your own environment.

```
# If you are running this for the first time you may have to install the mysql-connector libr
# If you are trying this on your own machine you will probably have to install it from the co
!pip install mysql-connector-python
```

```
Collecting mysql-connector-python
  Downloading https://files.pythonhosted.org/packages/6c/1d/e666f7d43496a2315d3963a2fb7
     |████████████████████████████████| 18.0MB 1.2MB/s
Requirement already satisfied: protobuf>=3.0.0 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (fro
Installing collected packages: mysql-connector-python
Successfully installed mysql-connector-python-8.0.22
```

```
# Import some libraries
import mysql.connector
from mysql.connector import errorcode
import datetime
```

I created a MySQL database using remotemysql.com

- Here I create a few variables for establishing the connection to it
- You will need to change these for whatever database you create as this one is likely to disapper soon

```
user='tG0RZCQgdF'
password = '1kaF93m1OE'
host = 'remotemysql.com'
DB_NAME = 'tG0RZCQgdF'
```

Programmatically add a table to the database

- I already created most of the tables of the ap database we used extensively in class (using MySQL Workbench connected to the remote database)
- I droped the ***vendor_contacts*** table to allow use create it programmatically

```
# Specify the table we want to create (This code would allow us to specify several)
TABLES = {}
TABLES['vendor_contacts'] = (
    "CREATE TABLE `vendor_contacts` ("
    "  `vendor_id` INT PRIMARY KEY,"
    "  `last_name` varchar(50) NOT NULL,"
    "  `first_name` varchar(50) NOT NULL)")

# Open a connection to the database
cnx = mysql.connector.connect(user=user, password=password, host=host, database=DB_NAME)
cursor = cnx.cursor()

def create_database(cursor):
    try:
        cursor.execute(
            "CREATE DATABASE {} DEFAULT CHARACTER SET 'utf8'".format(DB_NAME))
    except mysql.connector.Error as err:
        print("Failed creating database: {}".format(err))
        exit(1)

# Specify the database to use
try:
    cursor.execute("USE {}".format(DB_NAME))
except mysql.connector.Error as err:
    print("Database {} does not exist.".format(DB_NAME))
    if err.errno == errorcode.ER_BAD_DB_ERROR:
        create_database(cursor)
        print("Database {} created successfully.".format(DB_NAME))
        cnx.database = DB_NAME
    else:
        print(err)
        exit(1)

# Create each of the specified tables
for table_name in TABLES:
    table_description = TABLES[table_name]
    try:
        print("Creating table {}: ".format(table_name), end='')
        cursor.execute(table_description)
    except mysql.connector.Error as err:
        if err.errno == errorcode.ER_TABLE_EXISTS_ERROR:
            print("already exists.")
        else:
            print(err.msg)
    else:
```

```
        print("OK")

cursor.close()
cnx.close()
```

```
     Creating table vendor_contacts: OK
```

Programmatically insert data into the vendor_contacts table

```
cnx = mysql.connector.connect(user=user, password=password, host=host, database=DB_NAME)
cursor = cnx.cursor()

# Specify structure of INSERT query
add_vendor_contact = ("INSERT INTO vendor_contacts "
               "(vendor_id, last_name, first_name) "
               "VALUES (%s, %s, %s)")

# Create a list of values to INSERT
vendor_contact_list = [(5,'Davison','Michelle')\
                       , (12,'Mayteh','Kendall')\
                       , (17,'Onandonga','Bruce')\
                       ,(44,'Antavius','Anthony')\
                       , (76,'Bradlee','Danny')\
                       , (94,'Suscipe','Reynaldo')\
                       , (101,'O''Sullivan','Geraldine')\
                       , (123,'Bucket','Charles')]

# Insert each of the vendor contacts
for contact in vendor_contact_list:
  cursor.execute(add_vendor_contact, contact)

# Make sure data is committed to the database
cnx.commit()

cursor.close()
cnx.close()
```

Programatically Querying Data

```
# Open a connection to the database
cnx = mysql.connector.connect(user=user, password=password, host=host, database=DB_NAME)
cursor = cnx.cursor()

#  Filtering rows includes all rows between given dates
query = ("SELECT invoice_number, invoice_date, invoice_total "
        "FROM invoices "
        "WHERE invoice_date BETWEEN %s AND %s")

date_start = datetime.date(2018, 6, 1)
```

```
date_start = datetime.date(2018, 6, 1)
date_end = datetime.date(2018, 6, 30)

cursor.execute(query, (date_start, date_end))

for (invoice_number, invoice_date, invoice_total) in cursor:
  print("Invoice {} dated {:%d %b %Y}, was for ${}".format(invoice_number, invoice_date, invo

cursor.close()
cnx.close()
```

```
Invoice 40318 dated 01 Jun 2018, was for $21842.00
Invoice 111-92R-10094 dated 01 Jun 2018, was for $19.67
Invoice 989319-437 dated 01 Jun 2018, was for $2765.36
Invoice 547481328 dated 03 Jun 2018, was for $224.00
Invoice 31359783 dated 03 Jun 2018, was for $1575.00
Invoice 1-202-2978 dated 03 Jun 2018, was for $33.00
Invoice 111-92R-10097 dated 04 Jun 2018, was for $16.33
Invoice 547479217 dated 07 Jun 2018, was for $116.00
Invoice 989319-477 dated 08 Jun 2018, was for $2184.11
Invoice Q545443 dated 09 Jun 2018, was for $1083.58
Invoice 111-92R-10092 dated 09 Jun 2018, was for $46.21
Invoice 97/553B dated 10 Jun 2018, was for $313.55
Invoice 963253245 dated 10 Jun 2018, was for $40.75
Invoice 367447 dated 11 Jun 2018, was for $2433.00
Invoice 75C-90227 dated 11 Jun 2018, was for $1367.50
Invoice 963253256 dated 11 Jun 2018, was for $53.25
Invoice 4-314-3057 dated 11 Jun 2018, was for $13.75
Invoice 989319-497 dated 12 Jun 2018, was for $2312.20
Invoice 24946731 dated 15 Jun 2018, was for $25.67
Invoice 963253269 dated 15 Jun 2018, was for $26.75
Invoice 989319-427 dated 16 Jun 2018, was for $2115.81
Invoice 963253267 dated 17 Jun 2018, was for $23.50
Invoice 509786 dated 18 Jun 2018, was for $6940.25
Invoice 263253253 dated 18 Jun 2018, was for $31.95
Invoice 989319-487 dated 20 Jun 2018, was for $1927.54
Invoice MABO1489 dated 21 Jun 2018, was for $936.93
Invoice 133560 dated 22 Jun 2018, was for $175.00
Invoice 24780512 dated 22 Jun 2018, was for $6.00
Invoice 963253254 dated 22 Jun 2018, was for $108.50
Invoice 43966316 dated 22 Jun 2018, was for $10.00
Invoice CBM9920-M-T77109 dated 23 Jun 2018, was for $290.00
Invoice 109596 dated 24 Jun 2018, was for $41.80
Invoice 7548906-20 dated 24 Jun 2018, was for $27.00
Invoice 963253248 dated 24 Jun 2018, was for $241.00
Invoice 97/553 dated 25 Jun 2018, was for $904.14
Invoice 97/522 dated 28 Jun 2018, was for $1962.13
Invoice 587056 dated 30 Jun 2018, was for $2184.50
```

Here might be a good example of accessing data from an open NoSQL database with GIS data if you want to explore further

https://towardsdatascience.com/using-geotabs-open-datasets-visualizing-results-using-python-and-colab-notebooks-1657cb50d099