BIOINF 575 HW 07: SQL

Problem #1: Python Package Exploration

A. I will be exploring the package Mysql-Connector-Python. I found this package through the Anaconda link provided in the HW_07 PDF, under the packages category. Due to its long name, I'll be referring to it as mysql from now on.

B. I selected this package because I really enjoyed the SQL lectures in class, and because I may need/would like to use SQL to build specific databases to organize possibly my own generated biological data (due to my wet lab background). So this package will help me explore more in-depth SQL.

C. GitHub/mysql-connector-python/setup.py -> Mysql requires Python versions 3, 3.6, 3.7, 3.8, and 3.9.

D. GitHub/mysql-connector-python/setup.py -> When searching for "Operating System", I get "OS Independent" for mysql.

E. GitHub/mysql-connector-python/setup.py -> Mysql requires the package called protobuf, version 3.0.0.

F. Class chosen:

```
1. The class I chose is called DummySocket(object). It is found through
this pathway: mysql-connector-python/tests/__init___.py/ and is located
on line 173 in the code.

2. The class DummySocket tests socket connection (I looked this up and
from my understanding, a socket is one endpoint of a two-way
communication link between two programs running on the same network)
without generating network activity (I looked this up too and I learned
it means the amount of data moving across a computer network at any given
time). The documentation stated it is a proxy class, which I looked up
and learned that it receives client requests, performs work, and then
passes the request to a service object.

3. An object of this type could be used mainly for testing purposes
before using real service objects by clients that would lead to network
activity. My understanding could be wrong, but maybe this class can
perform certain tests on requests, making sure those requests are
filtered or pass certain criteria before being passed to a service
object.

4. Yes, it appears DummySocket inherits from one class called "Object".

5. To DummySocket, I would add the functionality of being able to check
if the tables in the database do indeed contain the correct type of data
that I want. If they do not contain the correct data, I want a specific
```

```
error to be raised informing me that I must change such data type to
avoid future problems when trying to do querys on the database. It will
save a lot of headache I think!
```

In [ ]:

Problem #2: SQL

Part A: Creating SQL Database, Tables, and Indexes

In [147…

```
#create the database.
from sqlite3 import connect
connection = connect("worm_genome.sqlite")
cursor = connection.cursor()
```

In [148…

```
#create features table.
#I took the error handeling code from class session 19 (Exception Handling).

create_features = '''
CREATE TABLE IF NOT EXISTS features (
    feature_id INTEGER PRIMARY KEY AUTOINCREMENT,
    seq_id TEXT NOT NULL,
    source TEXT NOT NULL,
    type_id TEXT NOT NULL,
    start BIGINT NOT NULL,
    end BIGINT NOT NULL,
    score TEXT NOT NULL, --score REAL (no need to include NOT NULL)
    strand TEXT NOT NULL,
    phase TEXT NOT NULL
    );
'''

try:
    cursor.execute(create_features)
except connection.DatabaseError:
    print("Creating the features table resulted in a database error!")
    connection.rollback()
    raise
else:
    connection.commit()
finally:
    print("done!")
```

```
done!
```

In [149…

```
#checking to ensure my features table was created in the database.
select_master = "SELECT name, type FROM sqlite_master;"
cursor.execute(select_master)
cursor.fetchall()
```

Out[149…

```
[('features', 'table'), ('sqlite_sequence', 'table')]
```

In [150…

```
#create attributes table.
```

```
#I took the error handeling code from class session 19 (Exception Handling).

create_attributes = '''
CREATE TABLE IF NOT EXISTS attributes (
    attr_id INTEGER PRIMARY KEY AUTOINCREMENT,
    feature_id INTEGER NOT NULL,
    attr_name TEXT NOT NULL,
    value TEXT NOT NULL,
    FOREIGN KEY (feature_id) REFERENCES features (feature_id)
    );
'''

try:
    cursor.execute(create_attributes)
except connection.DatabaseError:
    print("Creating the attributes table resulted in a database error!")
    connection.rollback()
    raise
else:
    connection.commit()
finally:
    print("done!")
```

```
done!
```

In [151…

```
#checking to ensure my attributes table was created in the database.
cursor.execute(select_master)
cursor.fetchall()
```

Out[151…

```
[('features', 'table'), ('sqlite_sequence', 'table'), ('attributes', 'table')]
```

In [152…

```
#creating the indexes for features table.
create_feat_type_index = '''
CREATE INDEX type_idx
ON features (type_id);
'''
cursor.execute(create_feat_type_index)
connection.commit()
```

In [153…

```
create_start_index = '''
CREATE INDEX start_idx
ON features (start);
'''
cursor.execute(create_start_index)
connection.commit()
```

In [154…

```
create_end_index = '''
CREATE INDEX end_idx
ON features (end);
'''
cursor.execute(create_end_index)
connection.commit()
```

In [155…

```
#creating the indexes for attributes table.
```

```
create_feature_id_index = '''
CREATE INDEX feature_id_idx
ON attributes (feature_id);
'''
cursor.execute(create_feature_id_index)
connection.commit()
```

In [156…

```
create_attr_name_index = '''
CREATE INDEX attr_name_idx
ON attributes (attr_name);
'''
cursor.execute(create_attr_name_index)
connection.commit()
```

In [157…

```
#checking both tables and all indexes that I just created are in my database, which the
cursor.execute(select_master)
cursor.fetchall()
```

Out[157…

```
[('features', 'table'),
 ('sqlite_sequence', 'table'),
 ('attributes', 'table'),
 ('type_idx', 'index'),
 ('start_idx', 'index'),
 ('end_idx', 'index'),
 ('feature_id_idx', 'index'),
 ('attr_name_idx', 'index')]
```

In [ ]:

Part B: Populating the Database

In [158…

```
#inserting the data from the gff file into both features and attributes tables.
#I implemented code from study session 11 and class session 23 to parse and populate th
#the insert commands are executed while parsing the gff file.
#commented lines besides those describing the line of code are other ways I could parse
#that I explored with Cristina. They are there for future reference only.

insert_features = '''
INSERT INTO features (seq_id, source, type_id, start, end, score, strand, phase)
VALUES (?,?,?,?,?,?,?,?); --how many values we want to insert. Template/placeholder.
'''

insert_attributes = '''
INSERT INTO attributes (feature_id, attr_name, value)
VALUES (?,?,?);
'''

#parsing the gff file below:
#opening the file to read the input data for tables.
with open("worm_genome.gff3") as worm_genome_file:
    #going through each line in the file.
    for line in worm_genome_file:
        #disregard the lines beginning with #'s.
        if not line.startswith("#"):
            #split the lines by tab. Not sure about strip
            #so maybe strip is not needed/extra.
            line_elems = line.strip().split("\t")
            #assign each column in my features table to the respective input from the f
            seq_id = line_elems[0]
            source = line_elems[1]
            type_id = line_elems[2]
            start = line_elems[3]
#             start = int(line_elems[3])
#             if line_elems[3] == ".":
#                 start = -1
#             else:
#                 start = int(line_elems[3])
            end = line_elems[4]
            score = line_elems[5]
#             if line_elems[5] == ".":
#                 score = "NULL"
#             else:
#                 score = float(line_elems[5])
            strand = line_elems[6]
            phase = line_elems[7]
            #setting all these variables equal to one variable.
            features_data = (seq_id, source, type_id, start, end, score, strand, phase)
            #SQL to insert features_data into features table. Insert command above.
            cursor.execute(insert_features, features_data)

            #Using lastrowid to return the value generated from the autoincrement
            #column in the features table. Needed because we need feature_id to be incl
            feature_id = cursor.lastrowid
            #take the last element in the line_elems list. That will contain the data f
            attr_str = line_elems[-1]
            #go through each pair of attributes and split by ";".
            for pair in attr_str.split(";"):
                #splitting the pair by name and value_name
                name, values = pair.split("=")
                #go through each element in the value list and split by ",".
                for value_name in values.split(","):
                    attr_name = name
                    value = value_name
                    attributes_data = (feature_id, attr_name, value)
                    cursor.execute(insert_attributes, attributes_data)

                    #break
```

In [171…

```
select_features = "SELECT * FROM features LIMIT 20;"
cursor.execute(select_features)
cursor.fetchall()
```

Out[171…

```
[(1, 'I', 'WormBase', 'chromosome', 1, 15072434, '.', '.', '.'),
 (2, 'I', 'WormBase', 'ncRNA_gene', 3747, 3909, '.', '-', '.'),
 (3, 'I', 'WormBase', 'snoRNA', 3747, 3909, '.', '-', '.'),
 (4, 'I', 'WormBase', 'exon', 3747, 3909, '.', '-', '.'),
 (5, 'I', 'WormBase', 'gene', 4116, 10230, '.', '-', '.'),
 (6, 'I', 'WormBase', 'mRNA', 4116, 10230, '.', '-', '.'),
 (7, 'I', 'WormBase', 'three_prime_UTR', 4116, 4220, '.', '-', '.'),
 (8, 'I', 'WormBase', 'exon', 4116, 4358, '.', '-', '.'),
 (9, 'I', 'WormBase', 'CDS', 4221, 4358, '.', '-', '0'),
 (10, 'I', 'WormBase', 'exon', 5195, 5296, '.', '-', '.'),
 (11, 'I', 'WormBase', 'CDS', 5195, 5296, '.', '-', '0'),
 (12, 'I', 'WormBase', 'exon', 6037, 6327, '.', '-', '.'),
 (13, 'I', 'WormBase', 'CDS', 6037, 6327, '.', '-', '0'),
 (14, 'I', 'WormBase', 'exon', 9727, 9846, '.', '-', '.'),
 (15, 'I', 'WormBase', 'CDS', 9727, 9846, '.', '-', '0'),
 (16, 'I', 'WormBase', 'CDS', 10095, 10148, '.', '-', '0'),
 (17, 'I', 'WormBase', 'exon', 10095, 10230, '.', '-', '.'),
 (18, 'I', 'WormBase', 'five_prime_UTR', 10149, 10230, '.', '-', '.'),
 (19, 'I', 'WormBase', 'gene', 11495, 16837, '.', '+', '.'),
 (20, 'I', 'WormBase', 'mRNA', 11495, 16793, '.', '+', '.')]
```

In [172…

```
select_attributes = "SELECT * FROM attributes LIMIT 20;"
cursor.execute(select_attributes)
cursor.fetchall()
```

Out[172…

```
[(1, 1, 'ID', 'chromosome:I'),
 (2, 1, 'Alias', 'BX284601.5'),
 (3, 1, 'Alias', 'NC_003279.8'),
 (4, 2, 'ID', 'gene:WBGene00023193'),
 (5, 2, 'Name', 'Y74C9A.6'),
 (6, 2, 'biotype', 'snoRNA'),
 (7, 2, 'gene_id', 'WBGene00023193'),
 (8, 2, 'logic_name', 'wormbase_non_coding'),
 (9, 3, 'ID', 'transcript:Y74C9A.6'),
 (10, 3, 'Parent', 'gene:WBGene00023193'),
 (11, 3, 'Name', 'Y74C9A.6'),
 (12, 3, 'biotype', 'snoRNA'),
 (13, 3, 'transcript_id', 'Y74C9A.6'),
 (14, 4, 'Parent', 'transcript:Y74C9A.6'),
 (15, 4, 'Name', 'Y74C9A.6.e1'),
 (16, 4, 'constitutive', '1'),
 (17, 4, 'ensembl_end_phase', '-1'),
 (18, 4, 'ensembl_phase', '-1'),
 (19, 4, 'exon_id', 'Y74C9A.6.e1'),
 (20, 4, 'rank', '1')]
```

In [173…

```
#need to commit before moving on to Query.
connection.commit()
```

In [ ]:

Part C: Query the Database

In [174…

```
#function to obtain the header taken from the class session 23 notebook.
def get_header(cursor):
    '''
    Makes a tab delimited header row from the cursor description.
    Arguments:
        cursor: a cursor after a select query
    Returns:
        string: A string consisting of the column names separated by tabs, no new line
    '''
    return '\t'.join([row[0] for row in cursor.description])
```

In [175…

```
#function to obtain the results taken from the class session 23 notebook.
def get_results(cursor):
    '''
    Makes a tab delimited table from the cursor results.
    Arguments:
        cursor: a cursor after a select query
    Returns:
        string: A string consisting of the column names separated by tabs, no new line
    '''
    res = list()
    for row in cursor.fetchall():
        res.append('\t'.join(list(map(str,row))))
    return "\n".join(res)
```

In [176…

```
#Query #1: Count the number of rows in the features table.
count_rows_features = "SELECT count(*) FROM features;"
cursor.execute(count_rows_features)
print(get_header(cursor))
print(get_results(cursor))
```

```
count(*)
659621
```

In [177…

```
#Query #2: Count the number of rows in the attributes table.
count_rows_attributes = "SELECT count(*) FROM attributes;"
cursor.execute(count_rows_attributes)
print(get_header(cursor))
print(get_results(cursor))
```

```
count(*)
3436229
```

In [178…

```
#Query #3: Display the type and number of features for each type from the features tabl
count_type_features = '''
SELECT DISTINCT type_id, count(feature_id)
FROM features
GROUP BY type_id;
'''
cursor.execute(count_type_features)
print(get_header(cursor))
print(get_results(cursor))
```

```
type_id count(feature_id)
CDS     222859
chromosome      7
exon    269904
five_prime_UTR  30960
gene    20222
lnc_RNA 283
mRNA    33391
miRNA   454
ncRNA   8402
ncRNA_gene      24765
piRNA   15364
pre_miRNA       257
pseudogene      1791
pseudogenic_transcript  1827
rRNA    22
snRNA   130
snoRNA  345
```

```
tRNA    634
three_prime_UTR 28004
```

In [179…

```
#Query #4: Display the seq_id and type from the features table and the attr_name and va
#for all the features with the type chromosome.
display_cols_both_tables = '''
SELECT seq_id, type_id, attr_name, value
FROM features feat
INNER JOIN attributes att ON feat.feature_id = att.feature_id
WHERE type_id = 'chromosome';
'''
cursor.execute(display_cols_both_tables)
print(get_header(cursor))
print(get_results(cursor))
```

```
seq_id  type_id attr_name       value
I       chromosome      ID      chromosome:I
I       chromosome      Alias   BX284601.5
I       chromosome      Alias   NC_003279.8
II      chromosome      ID      chromosome:II
II      chromosome      Alias   BX284602.5
II      chromosome      Alias   NC_003280.10
III     chromosome      ID      chromosome:III
III     chromosome      Alias   BX284603.4
III     chromosome      Alias   NC_003281.10
IV      chromosome      ID      chromosome:IV
IV      chromosome      Alias   BX284604.4
IV      chromosome      Alias   NC_003282.8
MtDNA   chromosome      ID      chromosome:MtDNA
MtDNA   chromosome      Alias   X54252.1
MtDNA   chromosome      Alias   NC_001328.1
V       chromosome      ID      chromosome:V
V       chromosome      Alias   BX284605.5
V       chromosome      Alias   NC_003283.11
X       chromosome      ID      chromosome:X
X       chromosome      Alias   BX284606.5
X       chromosome      Alias   NC_003284.9
```

In [182…

```
#Query #5: Display the type from the features table and the number of attributes (using
#for each feature type that has more than 50 attributes.
display_type_number_of_attr = '''
SELECT type_id, count(att.feature_id)
FROM features feat
INNER JOIN attributes att ON feat.feature_id = att.feature_id
GROUP BY feat.type_id
HAVING count(att.feature_id) > 50;
'''
cursor.execute(display_type_number_of_attr)
print(get_header(cursor))
print(get_results(cursor))
```

```
type_id count(att.feature_id)
CDS     668577
exon    2159232
five_prime_UTR  30960
gene    111530
lnc_RNA 1415
mRNA    166955
miRNA   2270
```

```
ncRNA   42010
ncRNA_gene      123825
piRNA   76820
pre_miRNA       1285
pseudogene      8745
pseudogenic_transcript  8925
rRNA    110
snRNA   650
snoRNA  1725
tRNA    3170
three_prime_UTR 28004
```

```
In [ ]:
```