CS2316 - HOMEWORK 06: BUREAU OF LABOR AND STATISTICS SQL ANALYSIS

Purpose

In today's society, the ability to store and manage various data is a necessity for almost every company out there. The primary way these companies store their valuable data is by utilizing relational database management systems, or better known as RDMS. Structured Query Language, or SQL, is one of the most popular languages for relational database management systems that allows us to easily store and manipulate tabular data.

Skills

This assignment will give you practice querying data from a MySQL database and retrieving the information through Python in order to manipulate it. Students will also be exposed to the structure and usefulness of the pymysql library, as well as gain insight on how to bridge the gap between Python and MySQL.

Knowledge

The types of problems you will encounter working with data sets and attempting to get them into a consistent format is similar to that of industry professionals working with real-world data sets.

Important

1. Due Date: **04/06/2020 at 11:59 pm**
2. This homework is graded out of **100** points, but you can earn up to **120** points.
3. This is an individual assignment. You may collaborate with other students in this class. Collaboration means talking through problems, assisting with debugging, explaining a concept, etc. Students may only collaborate with fellow students currently taking CS 2316, the TA's and the lecturer. You should not exchange code or write code for others. For individual assignments, each student must turn in a unique program. Your submission must not be substantially similar to another student's submission.
4. For Help:
    - TA Helpdesk (Schedule posted on class website)
    - Email TA's or use Piazza Forums Notes
    - How to Think Like a Computer Scientist
    - [http://openbookproject.net/thinkcs/python/english3e/]
    - [https://pymysql.readthedocs.io/en/latest/]
5. Comment out or delete all your function calls. Only global variables, and comments are okay to be outside the scope of a function. When your code is run, all it should do is run without any errors.
6. Do not wait until the last minute to do this assignment in case you run into problems.
7. **Read the entire specifications document before starting this assignment.**
8. **IF YOUR CODE CANNOT RUN BECAUSE OF AN ERROR, IT IS A 0%**

CS2316 - HOMEWORK 06: BUREAU OF LABOR AND STATISTICS SQL ANALYSIS

Introduction

The goal of this homework is to enhance your understanding of MySQL database management system and SQL queries. You are asked to write a few functions, each of which correspond to an SQL query (more below). You should test the quereis out first on your own MySQL server, compare the output of your queries to the ones given, then when you have one or more of them working, upload the entire file (which must be named HW06.py) to GradeScope to see how well your code performs on the test cases we have created. You can submit the homework file as many times as you'd like before the deadline.

Understanding the Provided Files

We have provided you with the following files:

`combined_annuals.txt`
`combined_quarters.txt`
`ownership_titles.txt`
`industry_titles.txt`
`agglevel_titles.txt`
`blsSchema.sql`
`blsPopulate.py`
`HW06.py`

Make sure you have downloaded all of them and placed them in the same folder.

Understanding the Data Files

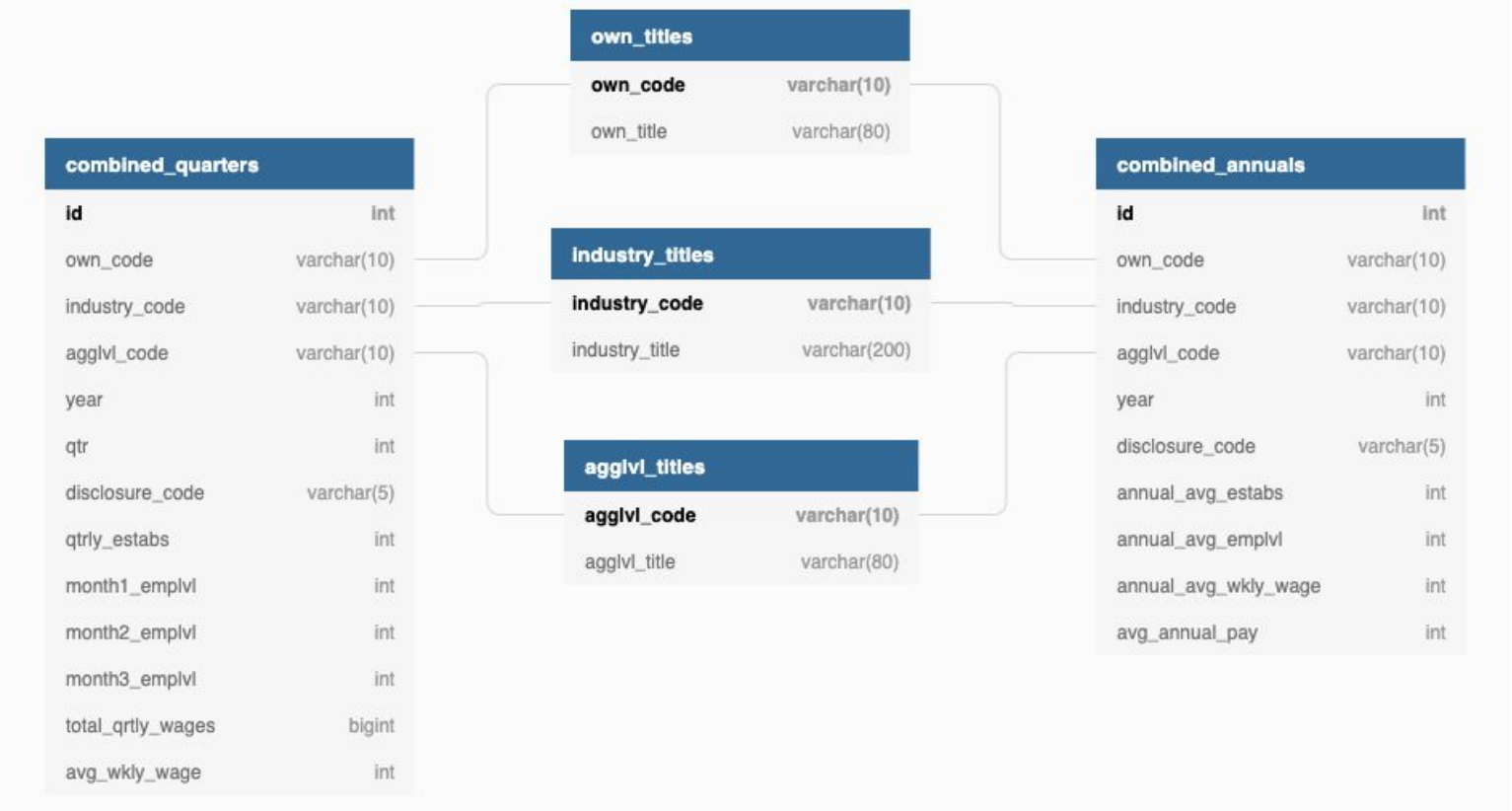
Data files for this assignment include:

`combined_annuals.txt`
`combined_quarters.txt`
`ownership_titles.txt`
`industry_titles.txt`
`agglevel_titles.txt`

If you open up any of the above txt files and observe its structure, you can see that they are actually CSV (Comma-Separated Value) files. However, it has the .txt extension rather than the .csv extension. We purposefully provided this CSV file in .txt format because files with the .csv extension often get converted to Excel spreadsheets automatically by computers, which would potentially mess up the original data.

The two files `combined_quarters.txt` and `combined_annuals.txt` consist of data downloaded from this website. Before you begin, you should become familiar with and understand the field names mentioned in the "Quarterly Data Slice Layout" and "Annual Average Data Slice Layout" tables, which are both located in the link mentioned previously. We specifically consolidated and filtered the data in two csv files, `combined_quarters.txt` and `combined_annuals.txt` that correspond to area_fips value of "US000" which is interpreted as the total U.S. The `combined_quarters.txt`

consists of quarters 1, 2, 3, and 4 for years 2016 and 2017, while the `combined_annuals.txt` file is an annual report for years 2016 and 2017. After inspecting the respective data layouts it is apparent that some of the field names (columns) are codes and need to be interpreted. We have provided you with the `ownership_titles.txt`, `industry_titles.txt`, and `agglevel_titles.txt` csv files containing the translations. These files translate the own_code, industry_code, and agglvl_code field names (columns) in `combined_quarters.txt` and `combined_annuals.txt`. Below is a diagram of our resulting blsQcew schema.



Each arrow represents a foreign key mapping from one table to the other. For example, consider the following partial tables. The own_code, industry_code, and agglvl_code columns reference their respective tables as a foreign key.

id	own_code	industry_code	agglvl_code	year	qtr
0	0	10	10	2016	1
1	1	10	11	2016	1
2	1	101	12	2016	1
3	1	1011	13	2016	1

combined_quarters.txt

CS2316 - HOMEWORK 06: BUREAU OF LABOR AND STATISTICS SQL ANALYSIS

own_code	own_title
0	Total Covered
1	Federal Government
2	State Government
3	Local Government

ownership_titles.txt

industry_code	industry_title
10	Total, all industries
101	Goods-producing
1011	Natural resources and mining
1012	Construction

industry_titles.txt

agglvl_code	agglvl_title
10	National, Total Covered
11	National, Total -- by ownership sector
12	National, by Domain -- by ownership sector
13	National, by Supersector -- by ownership sector

agglevel_titles.txt

Building the Database

You will use the following two files to build and populate the database:

- `blsSchema.sql`: the schema file of database blsQcew
- `blsPopulate.py`: a Python script that populates the database blsQcew using the data files

To build your own blsQcew database on your local machine, run the two files above by following the steps below respectively.

If you have a password to the MySQL server:

1. Open Terminal / Command Prompt and navigate to the directory where you saved all of your provided homework files
2. Run **`mysql -u root -p < blsSchema.sql`**
    a. Input the password for MySQL
3. Run **`python blsPopulate.py MySQLpassword`**
    a. Notice that you will pass in your MySQL password as a command line argument when running blsPopulate.py.
        i. For example, if your MySQL password is "2316", then you will run `python blsPopulate.py 2316`
4. Confirm that your tables within the blsQcew database have been populated

If you do **NOT** have a password to the MySQL server:

1. Open Terminal / Command Prompt and navigate to the directory where you saved all of your provided homework files
2. Run **`mysql -u root < blsSchema.sql`**
3. Run **`python blsPopulate.py ""`**
    a. Notice that you will pass in an empty string as a command line argument when running `blsPopulate.py`.
4. Confirm that your tables within the blsQcew database have been populated

When populating the database, you want to avoid inserting duplicate data into the database. For this reason, you should always run `blsSchema.sql`, which will give you a brand-new empty database, before running `blsPopulate.py`. Therefore, if for some reason, `blsPopulate.py` errors, make sure to rerun `blsSchema.sql` before running `blsPopulate.py` again.

FUNCTIONS FOR HW06.py

Note: The grading rubric at the end of the document for point distribution.

As mentioned above, you are asked to write a few Python functions, each of which corresponds to a query. Some of the functions will take in parameters that will be used to construct the queries. For each function, you should assign your final query to the variable `query` as one single string. We encourage you to use f-strings to incorporate the function parameters for the queries. However, you will not be penalized for not doing so.

Notice that we have provided you with these 3 lines of code in each function:

```
cursor.execute(query)
result = cursor.fetchall()
return result
```

Recall from lecture:

> 1st line: a pymysql cursor executes the query passed in.
> 2nd line: Because we are using the **pymysql.cursors.Cursor** cursorclass, the fetchall() method of the cursor object retrieves the query result from MySQL server and returns the result as ***a tuple of tuples***.
> 3rd line: return that tuple of tuples.

Do **NOT** change these 3 lines.

Do **NOT** hard code the resulting tuple of tuples from the query. We will manually check for hardcoding after the assignment is due and will deduct points accordingly.

In HW06.py, `query0()` has already been provided to you as an example.

CS2316 - HOMEWORK 06: BUREAU OF LABOR AND STATISTICS SQL ANALYSIS

Function name: **query0**
Parameter(s): `cursor` (pymysql.cursors.Cursor)
Return Type: tuple
Description:
Write a function that performs the following query:
Select all columns from the own_titles table.
Your output should contain 2 columns: own_code and own_title.

SQL Test Case:
Your query output should look like:

```
+----------+-----------------------------------------------------+
| own_code | own_title                                           |
+----------+-----------------------------------------------------+
|        0 | Total Covered                                       |
|        1 | Federal Government                                  |
|        2 | State Government                                    |
|        3 | Local Government                                    |
|        4 | International Government                            |
|        5 | Private                                             |
|        8 | Total Government                                    |
|        9 | Total U.I. Covered (Excludes Federal Government)    |
+----------+-----------------------------------------------------+
8 rows in set
```

Python Test Cases:

```
>>> cursor = create_cursor('localhost', 'root', user_password[1],
'blsQcew')
>>> query0(cursor)
(('0', 'Total Covered'),
 ('1', 'Federal Government'),
 ('2', 'State Government'),
 ('3', 'Local Government'),
 ('4', 'International Government'),
 ('5', 'Private'),
 ('8', 'Total Government'),
 ('9', 'Total U.I. Covered (Excludes Federal Government)'))
```

[1] Replace `user_password` with your MySQL password as a **string**. If you do not have a MySQL password set up, replace `user_password` with "" (empty string)

CS2316 - HOMEWORK 06: BUREAU OF LABOR AND STATISTICS SQL ANALYSIS

Function name: **`query1`**
Parameter(s): `cursor` (pymysql.cursors.Cursor)
Return Type: tuple
Description:
Write a function that performs the following query:
What is the average avg_annual_pay for all industries for year 2016 and year 2017, respectively?
Your output should contain 2 columns: year and avg(avg_annual_pay).

SQL Test Case:
Your query output should look like:

```
+------+----------------------+
| year | avg(avg_annual_pay) |
+------+----------------------+
| 2016 |          46918.3310 |
| 2017 |          48630.5649 |
+------+----------------------+
2 rows in set
```

Python Test Cases:

```
>>> cursor = create_cursor('localhost', 'root', user_password,
'blsQcew')
>>> query1(cursor)
((2016, Decimal('46918.3310')),
 (2017, Decimal('48630.5649')))
```

Function name: **`query2`**
Parameter(s): `cursor` (pymysql.cursors.Cursor), `year` (int), `lower_avg_annual_pay` (int), `upper_avg_annual_pay` (int)
Return Type: tuple
Description:
Write a function that performs the following query:
What are the industry titles whose average annual pays are between {lower_*avg_annual_pay*} and {*upper_avg_annual_pay*} in year {*year*}?
Both {lower_*avg_annual_pay*} and {*upper_avg_annual_pay*} are inclusive.
If there are duplicate industry titles, keep the duplicates. Sort the industry titles by the average annual pays in descending order.
Your output should contain 1 column: industry_title.

SQL Test Case:

If `year = 2016, lower_avg_annual_pay = 10000, upper_avg_annual_pay = 15000`, your query output should look like:

```
+-----------------------------------------------+
| industry_title                                |
+-----------------------------------------------+
| Drycleaning and laundry services              |
| Drycleaning and laundry services              |
| Traveler accommodation                        |
| Motion picture theaters, except drive-ins     |
| Motion picture and video exhibition           |
| Motion picture and sound recording industries |
| Motion picture and video industries           |
| Flight training                               |
| Amusement parks and arcades                   |
| ...                                           |
| Home health care services                     |
+-----------------------------------------------+
25 rows in set
```

Python Test Cases:

```
>>> cursor = create_cursor('localhost', 'root', user_password,
'blsQcew')
>>> query2(cursor, 2016, 10000, 15000)
(('Drycleaning and laundry services',),
 ('Drycleaning and laundry services',),
 ('Traveler accommodation',),
 ('Motion picture theaters, except drive-ins',),
 ('Motion picture and video exhibition',),
 ('Motion picture and sound recording industries',),
 ('Motion picture and video industries',),
 ('Flight training',),
 ('Amusement parks and arcades',),
 ...
 ('Home health care services',))
```

Function name: **`query3`**
Parameter(s): `cursor` (pymysql.cursors.Cursor), `keyword` (str)
Return Type: tuple
Description:

CS2316 - HOMEWORK 06: BUREAU OF LABOR AND STATISTICS SQL ANALYSIS

Write a function that performs the following query:
What are all the *unique* industry titles that contain the word "{*keyword*}"? Remember that since MySQL is case-insensitive, you do not need to worry about case sensitivity of the keyword or the industry titles. Sort the industry titles in ascending alphabetical order. Your output should contain 1 column: industry_title.

SQL Test Case:
If `keyword = 'engineering'`, your query output should look like:

```
+-----------------------------------------------------------------------+
| industry_title                                                        |
+-----------------------------------------------------------------------+
| AICS02 541710 Physical, engineering and biological research           |
| Architectural and engineering services                                |
| Engineering services                                                  |
| Heavy and civil engineering construction                              |
| Research and development in the physical, engineering, and life
sciences                                                               |
| Research and development in the physical, engineering, and life
sciences (except nanotechnology and biotechnology)                     |
+-----------------------------------------------------------------------+
6 rows in set
```

Python Test Cases:

```
>>> cursor = create_cursor('localhost', 'root', user_password,
'blsQcew')
>>> query3(cursor, 'engineering')
(('AICS02 541710 Physical, engineering and biological
research',),
 ('Architectural and engineering services',),
 ('Engineering services',),
 ('Heavy and civil engineering construction',),
 ('Research and development in the physical, engineering, and
 life sciences',),
 ('Research and development in the physical, engineering, and
 life sciences (except nanotechnology and biotechnology)',))
```

Function name: **`query4`**
Parameter(s): `cursor` (pymysql.cursors.Cursor), `year` (int), `quarter` (int)
Return Type: tuple
Description:
Write a function that performs the following query:
What are all the ownership titles and their respective average weekly wages during quarter {*quarter*} of year {*year*}? After finding all of the multiple corresponding avg_wkly_wage for each ownership title, you

should then take the average of those wages. Sort the output by the resulting average wages in descending order.
Your output should contain 2 columns: own_title and avg(avg_wkly_wage).

SQL Test Case:
If `year = 2017`, `quarter = 2`, your query output should look like:

```
+-------------------------------------------------+-------------------+
| own_title                                       | avg(avg_wkly_wage)|
+-------------------------------------------------+-------------------+
|Private                                          |         1125.7276 |
|Federal Government                               |         1115.0362 |
|Total Government                                 |         1077.0000 |
|Total Covered                                    |         1020.0000 |
|Total U.I. Covered (Excludes Federal Government)|         1010.0000 |
|State Government                                 |          537.5008 |
|Local Government                                 |          537.4628 |
+-------------------------------------------------+-------------------+
7 rows in set
```

Python Test Cases:

```
>>> cursor = create_cursor('localhost', 'root', user_password,
'blsQcew')
>>> query3(cursor, 2017, 2)
(('Private', Decimal('1125.7276')),
 ('Federal Government', Decimal('1115.0362')),
 ('Total Government', Decimal('1077.0000')),
 ('Total Covered', Decimal('1020.0000')),
 ('Total U.I. Covered (Excludes Federal Government)',
 Decimal('1010.0000')),
 ('State Government', Decimal('537.5008')),
 ('Local Government', Decimal('537.4628')))
```

Function name: **query5**
Parameter(s): `cursor` (pymysql.cursors.Cursor), `year` (int), `quarter` (int)
Return Type: tuple
Description:
Write a function that performs the following query:
What is the name of the industry that has the highest total employment in quarter *{quarter}* of year *{year}* and what is its total employment? You should exclude the industries whose industry codes = 10 from the output. Sort the output by the total employment level in descending order.
Your output should contain 2 columns: industry_title and total_emplvl.

Hint: to get the total_emplvl column, you will have to add up columns month1_emplvl, month2_emplvl, and month3_emplvl and rename the resulting column to total_emplvl. Refer to this link if you need additional help.

SQL Test Cases:

If `year = 2017`, `quarter = 1`, your query output should look like:

```
+-------------------+--------------+
| industry_title    | total_emplvl |
+-------------------+--------------+
| Service-providing |    297906824 |
+-------------------+--------------+
1 row in set
```

Note: It is okay if you get the "WARNING: truncated incorrect" message on GradeScope if your test case passes.

Python Test Cases:

```
>>> cursor = create_cursor('localhost', 'root', '', 'blsQcew')
>>> query3(cursor, 2017, 1)
(('Service-providing', 297906824),)
```

Function name: **query6**

Parameter(s): `cursor` (pymysql.cursors.Cursor), `year` (int), `quarter` (int), `avg_wkly_wage_lb` (int)

Return Type: tuple

Description:

Write a function that performs the following query:

For quarter *{quarter}* in year *{year}*, what are the agglvl_titles whose lowest average weekly wage is greater than *{avg_wkly_wage_lb}*? Note that there can be multiple of the same titles within agglvl_title. Your output should contain 2 columns: agglvl_title and min(avg_wkly_wage).

SQL Test Cases:

If `year = 2017`, `quarter = 1`, `avg_wkly_wage_lb = 1000`, your query output should look like:

```
+---------------------------+--------------------+
| agglvl_title              | min(avg_wkly_wage) |
+---------------------------+--------------------+
| National, Total Covered   |               1111 |
| Total Government (U.S.)   |               1057 |
| Total U.I. Covered (U.S.) |               1103 |
+---------------------------+--------------------+
3 rows in set (0.03 sec)
```

CS2316 - HOMEWORK 06: BUREAU OF LABOR AND STATISTICS SQL ANALYSIS

Python Test Cases:

```
>>> cursor = create_cursor('localhost', 'root', '', 'blsQcew')
>>> query3(cursor, 2017, 1, 1000)
(('National, Total Covered', 1111),
 ('Total Government (U.S.)', 1057),
 ('Total U.I. Covered (U.S.)', 1103))
```

Function name: **query7**
Parameter(s): `cursor` (pymysql.cursors.Cursor)
Return Type: tuple
Description:
Write a function that performs the following query:
What are all the possible ways to combine the "Federal Government" ownership title with all the other agglvl_titles? Return the first 10 rows.
Your output should contain 2 columns: own_title and agglvl_title.

SQL Test Cases:
Your query output should look like:

```
+--------------------+-----------------------------------------------------+
| own_title          | agglvl_title                                        |
+--------------------+-----------------------------------------------------+
| Federal Government | National, Total Covered                             |
| Federal Government | National, Total -- by ownership sector              |
| Federal Government | National, by Domain -- by ownership sector          |
| Federal Government | National, by Supersector -- by ownership sector     |
| Federal Government | National, NAICS Sector -- by ownership sector       |
| Federal Government | National, NAICS 3-digit -- by ownership sector      |
| Federal Government | National, NAICS 4-digit -- by ownership sector      |
| Federal Government | National, NAICS 5-digit -- by ownership sector      |
| Federal Government | National, NAICS 6-digit -- by ownership sector      |
| Federal Government | National, Private, total, by establishment size class |
+--------------------+-----------------------------------------------------+
10 rows in set
```

Python Test Case:

```
>>> cursor = create_cursor('localhost', 'root', '', 'blsQcew')
>>> query7(cursor)
(('Federal Government', 'National, Total Covered'),
 ('Federal Government', 'National, Total -- by ownership
sector'),
 ('Federal Government', 'National, by Domain -- by ownership
sector'),
 ('Federal Government', 'National, by Supersector -- by
ownership sector'),
```

```
 ('Federal Government', 'National, NAICS Sector -- by ownership
sector'),
 ('Federal Government', 'National, NAICS 3-digit -- by ownership
sector'),
 ('Federal Government', 'National, NAICS 4-digit -- by ownership
sector'),
 ('Federal Government', 'National, NAICS 5-digit -- by ownership
sector'),
 ('Federal Government', 'National, NAICS 6-digit -- by ownership
sector'),
 ('Federal Government',
  'National, Private, total, by establishment size class'))
```

Function name: `query8`
Parameter(s): `cursor` (pymysql.cursors.Cursor)
Return Type: tuple
Description:
Write a function that performs the following query:
Select all the columns and an additional column named title_length that contains the length of each title in the own_title column from the own_titles table. Sort the output by the title_length in ascending order.
Your output should contain 3 columns: own_code, own_title, title_length
Hint: To learn about how to calculate the length of a character sequence, you might find this link helpful.
SQL Test Cases:
Your query output should look like:

```
+----------+--------------------------------------------------+--------------+
| own_code | own_title                                        | title_length |
+----------+--------------------------------------------------+--------------+
| 5        | Private                                          |            7 |
| 0        | Total Covered                                    |           13 |
| 2        | State Government                                 |           16 |
| 3        | Local Government                                 |           16 |
| 8        | Total Government                                 |           16 |
| 1        | Federal Government                               |           18 |
| 4        | International Government                         |           24 |
| 9        | Total U.I. Covered (Excludes Federal Government) |           48 |
+----------+--------------------------------------------------+--------------+
8 rows in set
```

Python Test Case:

```
>>> cursor = create_cursor('localhost', 'root', '', 'blsQcew')
>>> query8(cursor)
```

```
(('5', 'Private', 7),
 ('0', 'Total Covered', 13),
 ('2', 'State Government', 16),
 ('3', 'Local Government', 16),
 ('8', 'Total Government', 16),
 ('1', 'Federal Government', 18),
 ('4', 'International Government', 24),
 ('9', 'Total U.I. Covered (Excludes Federal Government)', 48))
```

BONUS

There are **20** points of total bonus you can earn. You may attempt up to 2 queries for this section, with each worth up to **10** points. You must meet the following conditions in order to have your queries be eligible for bonus points.

Function name: **`query9`**
Parameter(s): `cursor` (pymysql.cursors.Cursor), *<other parameters>*
Return Type: tuple
Conditions:

1. Function must not error
2. Each function must answer a question that is practical and meaningful
3. The question must be thoroughly explain why the query is practical and meaningful and what the query is answering as a comment above the function
4. Each function must include at least 2 additional parameters, which are incorporated to the query
5. Must join at least 2 tables
6. Must include conditionals (where or having)
7. Provide a test case

Function name: **`query10`**
Parameter(s): `cursor` (pymysql.cursors.Cursor), *<other parameters>*
Return Type: tuple
Conditions:

1. Function must not error
2. Each function must answer a question that is practical and meaningful
3. The question must be thoroughly explain why the query is practical and meaningful and what the query is answering as a comment above the function
4. Each function must include at least 2 additional parameters, which are incorporated to the query
5. Must join at least 2 tables
6. Must include conditionals (where or having)
7. Provide a test case

CS2316 - HOMEWORK 06: BUREAU OF LABOR AND STATISTICS SQL ANALYSIS

Gradescope/Canvas Requirements

- **NO PRINT STATEMENTS** this will break the autograder
- **NO FUNCTION CALLS** outside of function definitions this will also break the autograder
- Make sure the file submitted is titled **HW06.py**
- **Allowed imports: pymysql** and **pprint**
- **Only submit file HW06.py**

Grading Rubric

query1	10	pts
query2	10	pts
query3	10	pts
query4	10	pts
query5	15	pts
query6	15	pts
query7	15	pts
query8	15	pts
query9 (bonus)	10	pts
query10 (bonus)	10	pts
Total	120/100	pts