Practice Problems 5: Python Queries

1. Setup and Basics

For the first part of the practice problems you will need to install the MySQL connector module so your program can connect to your database server. Then, you will need to construct a simple program named **simpleDBC.py** that connects to the world database that we have previously used this semester, and work with the database cursor to execute queries and fetch and output results.

To complete this part of the practice problems:

Execute `pip install mysql-connector-python` in the Window's Command Prompt or MacOS Terminal

Create a new Python program called **simpleDBC.py** and inside:

- Import `mysql.connector`
- In a new variable, store the database connection of:

```
mysql.connector.connect(
host="localhost",
user="your username",
password="your password",
database="world")
```

  - Execute your program to test if the connection works, if it doesn't check the username and password
- In a new variable, create and store a new database cursor using the `cursor()` function with your database connection
- Print out all of the column names currently stored in your cursor, with a suitable message
  - Note this should be empty
- Execute the query "`select * from city`" by calling the `execute()` function from your cursor, and passing it the string query
- Print out all of the column names currently stored in your cursor, with a suitable message
  - The system may throw an error "Unread result found" if you execute with unread results in the cursor and that is ok at this point
- Print out the row count currently stored in your cursor, with a suitable message
  - Note this should be 0 at moment
- Fetch all of the results from the query and store them in a new variable using the `fetchall()` function from your cursor
- Print out the row count currently stored in your cursor, with a suitable message
- Using a `for` loop, iterate through your results printing out each record on a new line
- Close the cursor
- Close the database connection

2. Complex Results and Processing

Now that you have the basics setup, let's look at some more interesting ways we can execute queries and process results.

Create a new Python program called **advDBC.py** and inside:

- Import any necessary libraries
- Create and store the database connection to the world database
- Create and store a new cursor to the database
- Execute the query that would provide all of the attributes for all cities where the country code is AFG
    - Remember strings in queries still need to be between " ", so you may need to use the escape character \ to add necessary " "
- Fetch all of the results from the query and store them in a new variable
- Using a `for` loop, iterate through your results printing out just the name of each city on a new line
    - Remember that you can use the [ ] operators to access data from a particular attribute in a record (which is stored as a `Tuple`) if you know the position of the attribute in the table's schema
- Print out a newline
- Execute the same query again
- Fetch one record from the query and store it in a new variable using the `fetchone()` function from your cursor
- Using a `while` loop and print out just the name of each city on a new line and then fetch another record
    - Remember `fetchone()` returns `None` if there are no more records
- Print out a newline
- Execute the query that would provide all of the attributes for all cities where the country code is GGG
- Print the row count for this query, with an appropriate message
- Using a `for` loop, iterate through your results printing out just the name of each city on a new line
    - Notice how many results you retrieve this time!
- Print out a newline
- Close the current cursor
- Create a new cursor, storing it in the same variable, and have it use a dictionary structure instead of tuple structure, by setting the dictionary keyword argument to True
- Execute the query that would provide all of the attributes for all cities where the country code is AFG
- Using a `for` loop, iterate through your results printing out just the name of each city on a new line
    - Remember results are now stored as a dictionary, so if you want the results for a particular attribute you need to use its name in the [ ] operators
- Print out a newline
- Execute the query that would provide the average population and country code for each city, grouped by country code
- Print out all of the column names currently stored in your cursor, with a suitable message
- Print out a newline
- Execute the same query but rename the average population attribute to "avgPop"
- Print out all of the column names currently stored in your cursor
- Using a `for` loop, iterate through your results printing out the average population, with two decimal points of precision, and country code on a new line
    - Remember that format specifiers allow us to format Strings in Python, and most other languages.

- Close the cursor
- Close the database connection

3. Inserts, Deletes, and Updates

For the final part of the practice problems, you will need to work on inserting, deleting, and updating records using Python. Note that before changes made via queries are actually applied, the commit() function must be executed by the database connection. If that does not happen before the program ends, the changes will be lost!

Create a new Python program called **iduDBC.py** and inside:

- Import any necessary libraries
- Create and store the database connection to the characters database
- Create and store a new cursor to the database
- Execute a query that would select all attributes from the npcs table
- Using a `for` loop, iterate through your results printing out each record on a new line
- Print out a newline
- Execute a query that would insert a new record with the following characteristics:
    - id is 11
    - first name is Breia
    - last name is De Marlo
    - race is gnome
    - age is 120
    - occupation is bard
    - alignment is good
- Commit the change using the `commit()` function with the database connection
- Print the cursor's rowcount to check if the query succeeded
    - It should be 1
- Execute a query that would select all attributes from the npcs table
- Using a `for` loop, iterate through your results printing out each record on a new line
- Print out a newline
- Execute a query that would delete npcs from that table if their id is 11
- Commit the change
- Print the cursor's rowcount to check if the query succeeded
    - It should be 1
- Execute a query that would select all attributes from the npcs table
- Using a `for` loop, iterate through your results printing out each record on a new line
- Print out a newline
- Create a variable that stores insert portion of the previous query and use format specifiers as placeholders for the values
    - Remember all of the format specifiers must be string types!
- Create a variable that stores a `Tuple` containing the values for the previous query
- Execute the query using both variables
- Commit the change
- Print the cursor's rowcount to check if the query succeeded
    - It should be 1
- Execute a query that would select all attributes from the npcs table
- Using a `for` loop, iterate through your results printing out each record on a new line

- Print out a newline
- Execute a query that would delete npcs from that table if their id is 11
- Commit the change
- Print the cursor's rowcount to check if the query succeeded
- Close the cursor
- Close the database

Submission

Once you have completed this set of practice problems, please submit the following files to the Practice Problems 5 assignment in Canvas:

- **simpleDBC.py**
- **advDBC.py**
- **iduDBC.py**