Laboratory Worksheet Seven

NOTE: *You should complete all previous worksheets before continuing*

NOTE: *This worksheet includes work necessary for Assignment 1*

In last week's practical we investigated how to create a Jupyter Notebook and establish a connection between it and a database from MS SQL SERVER. While we could see the results of our queries, the formatting was not the most readable or presentable. In this week's lab, we'll further investigate the combination of Python with SQL to create more presentable query results.

PART 1: REMINDER! CREATING A SELECT QUERY

- Open a new Jupyter Notebook.
- If you are on the lab machines, you will need to run the install commands from last week in the first cell:

  !pip install ipython-sql

  !pip install pyodbc
- Remember to **wait** until the cell no longer has 'In[*]' next to it because the '*' indicates the cell is still running.
- As we are running Python in the notebook, we need to remember to include any necessary imports at the beginning of a new notebook. In the next cell type and then run the following code:

  import pandas as pd

  import pyodbc

- Next, write the Python and SQL code that will connect to the database, create and execute a **SELECT** query and finally display the result. Your cell should look similar to the screenshot in the next page.
  - **TIME SAVER:** You can avoid having to specify *projemp.dbo* before every table name by including *Database=projemp* in the String used to establish the connection with the database.
  - **Remember**, if you are on your own computer, you will likely not need the ***uid*** or ***pwd*** fields in the connection string.

```
conn = pyodbc.connect("Driver={SQL Server};Server=localhost\SQLEXPRESS;Database=projemp;uid=sa;pwd=Labuser1")
cursor = conn.cursor()
selectQuery = """
SELECT *
FROM emp
"""
result = cursor.execute(selectQuery)
for row in result:
    print(f'{row}')
conn.close()
```

- **NOTE:** When assigning a String literal to a variable, such as *selectQuery*, we can encapsulate the value of the String literal within **two pairs of three quotation mark characters (“““)**, therefore, allowing us to write the String literal on multiple lines instead of just one. This should help make the SQL more readable.

PART 2: DISPLAYING QUERY RESULT AS A TABLE

- So far, we have been printing the result of a **SELECT** query by using *print(f'{row}')* within a FOR LOOP. This is useful for viewing the entirety of the query result.
- To be able to view specific rows, we can create a Python list, and append each row of the result table into it.
- Now, copy the Python code from the current cell into a new cell.
    - Update the code in the new cell so that it looks like the screenshot below (the boxes show the updates that need to be made):

```
conn = pyodbc.connect("Driver={SQL Server};Server=localhost\SQLEXPRESS;Database=projemp;uid=sa;pwd=Labuser1")
cursor = conn.cursor()
selectQuery = """
SELECT *
FROM emp
"""
result = cursor.execute(selectQuery)
resultList = []
for row in result:
    resultList.append(row)

print(resultList)
conn.close()
```

- **Run** the code**.**
- Currently, you’ll get the same output from this code as you did in the previous cell.
- Since the rows have now been appended to the Python list *resultList*, we can access specific positions from the list, which means we can access specific rows from the result table.
- Update the *print* statement in the cell to *print(resultList[0])*

- The addition of *[0]* to the *print* statement means that we are now accessing the row stored at position 0 of *resultList*, which is the first row of the **SELECT** query's result table (lists' and arrays' first position is always indexed with 0).

- **Run** the code.
- You should now see only the employee record for Armstrong in the cell output.
  - The *0* in *resultList[0]* can be changed to any number from 0 to the total number of employees in the **EMP** table (by default the total number is 21 so numbers in the range 0-20 can be used inside the square brackets).
- Next, we can consider how to make updates to the code so that the query result is displayed as a formatted table.
- Now, copy the code into a new cell.
  - Update the code in the new cell so that it looks like the screenshot below (the boxes show the updates that need to be made):

```
conn = pyodbc.connect("Driver={SQL Server};Server=localhost\SQLEXPRESS;Database=projemp;uid=sa;pwd=Labuser1")
cursor = conn.cursor()
selectQuery = """
SELECT *
FROM emp
"""
result = cursor.execute(selectQuery)
resultList = []
for row in result:
    row = list(row)
    resultList.append(row)

tableColumns = ["Employee Number", "Employee Name", "Salary", "Age", "Supervisor Number", "Department Number"]
resultTable = pd.DataFrame(resultList, columns = tableColumns)
display(resultTable)
conn.close()
```

  - The inclusion of the line *row = list(row)* converts the current row to become a list itself, meaning that each index of the row list will be one of the column values.
  - The row list is then appended to *resultList.* Due to the **FOR LOOP**, this means that every row from the query result table will be converted to a list. As a result, once the loop finishes, *resultList* will be a list consisting of lists. This is important as *pd.DataFrame* requires its first parameter to be a list of lists.
  - The *tableColumns* list is a list of String literals where each String literal is a column name of the final result table.
  - The built-in function *pd.DataFrame* is then used to convert the list of lists (*resultList*) into a data frame (which is essentially a formatted table). The *tableColumns* list previously defined is used as the value for the second *pd.DataFrame* parameter.

- The result of the built-in function is then saved into the variable *resultTable.*
- At the beginning of the program, we included the line *import pandas as pd* so that we could use the *pd.DataFrame* function (where pd is just an alias for the *pandas* library name).

- **Run the code**. The output should look similar to the screenshot below:

	Employee Number	Employee Name	Salary	Age	Supervisor Number	Department Number
0	e1	armstrong	50000.00	56	None	d1
1	e10	jones	50000.00	49	e1	d3
2	e11	kelly	50000.00	36	e7	d2
3	e12	mccoy	50000.00	29	e3	d2
4	e13	neeson	50000.00	36	e19	d1
5	e14	pearson	50000.00	35	e17	d3
6	e15	pearse	50000.00	28	e21	d1
7	e16	quinn	50000.00	54	e2	d1
8	e17	roberts	50000.00	27	e4	d3
9	e18	smyth	50000.00	34	e21	d3
10	e19	trainor	50000.00	29	e7	d1

- Now, we have a more presentable and readable output from the query.

PART 3: COMBINING QUERIES WITHIN ONE CELL

- When executing queries that don't have a direct output e.g. **INSERT INTO, UPDATE** and **DELETE**, it requires the follow up process of checking the database to verify the query has had its intended effect.
- It can useful, therefore, to include a **SELECT** query, to be able to immediately demonstrate the effect of an **INSERT INTO, UPDATE** or **DELETE** query on a table.
- Start a new cell.
  - Add the code shown in the screenshot at the top of the next page (there is considerably more code than in previous cells, so comments have been included to help categorise blocks of code):

```
# Establish connection with database
conn = pyodbc.connect("Driver={SQL Server};Server=localhost\SQLEXPRESS;Database=projemp;uid=sa;pwd=Labuser1")
cursor = conn.cursor()
############################################################

# Execute and commit INSERT INTO query
insertQuery = """
INSERT INTO emp (eno, ename, salary, age, supno, dno)
VALUES (?, ?, ?, ?, ?, ?);
"""
values = ['e23', 'Corr', 55000.00, 21, 'e4', 'd1']
cursor.execute(insertQuery, values)
conn.commit()
############################################################

# Execute SELECT query
selectQuery = """
SELECT *
FROM emp
WHERE eno = ?
"""
enoValue = 'e23'
result = cursor.execute(selectQuery, enoValue)
############################################################

# Handle and display output of SELECT query
resultList = []
for row in result:
    row = list(row)
    resultList.append(row)

tableColumns = ["Employee Number", "Employee Name", "Salary", "Age", "Supervisor Number", "Department Number"]
resultTable = pd.DataFrame(resultList, columns = tableColumns)
display(resultTable)
############################################################

# Close connection to the database
conn.close()
```

- In this cell, we are inserting a new record into the **EMP** table and then immediately verifying that the operation has had the desired effect by including a **SELECT** query which queries the database for the record that has just been inserted. This process combines techniques introduced in the previous practical with the use of the *pd.DataFrame* function.
- **Run the code.** The output should look like the screenshot below:

	Employee Number	Employee Name	Salary	Age	Supervisor Number	Department Number
0	e23	Corr	55000.00	21	e4	d1

- Now, try the same process with **UPDATE** and **DELETE** queries of your choice so that you can verify the result of the queries immediately after executing them.