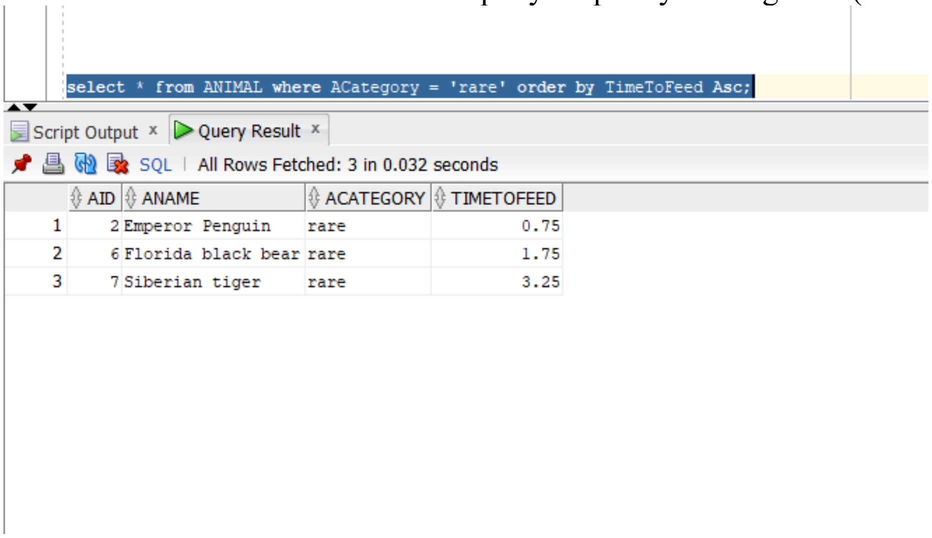
Name: Syed Noor Razi Ali StudentID: 2070326

DSC 450: Database Processing for Large-Scale Analytics
Assignment Module 4

Part 1

A) Using the extended Zoo database (ZooDatabase_extended.sql), write the following queries in SQL:

1. Find all the rare animals and sort the query output by feeding time (from small to large)

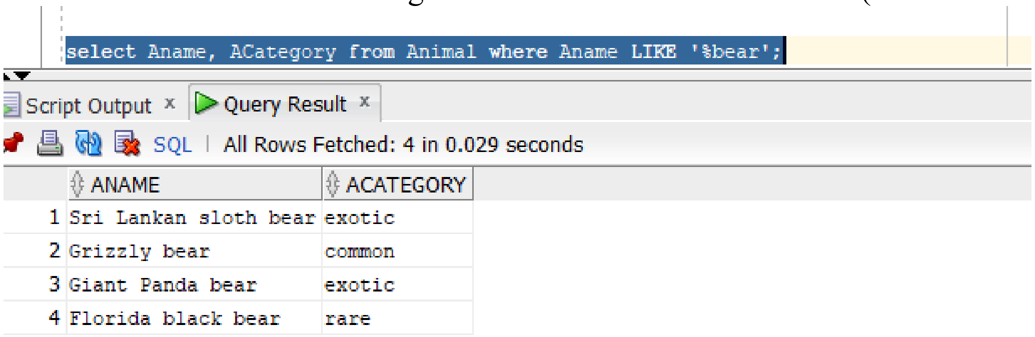


```
select * from ANIMAL where ACategory = 'rare' order by TimeToFeed Asc;
```

All Rows Fetched: 3 in 0.032 seconds

	AID	ANAME	ACATEGORY	TIMETOFEED
1	2	Emperor Penguin	rare	0.75
2	6	Florida black bear	rare	1.75
3	7	Siberian tiger	rare	3.25

2. Find the animal names and categories for animals related to a bear (hint: use the LIKE operator)

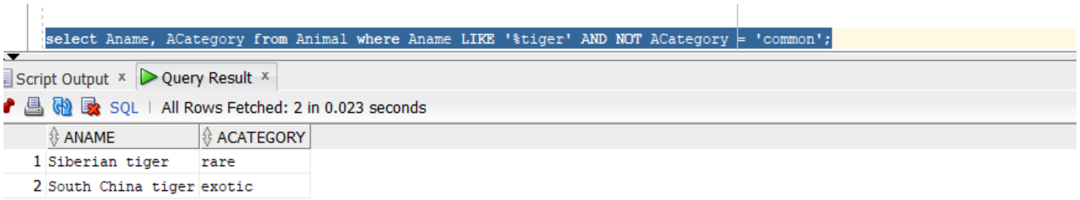


```
select Aname, ACategory from Animal where Aname LIKE '%bear';
```

All Rows Fetched: 4 in 0.029 seconds

	ANAME	ACATEGORY
1	Sri Lankan sloth bear	exotic
2	Grizzly bear	common
3	Giant Panda bear	exotic
4	Florida black bear	rare

3. Find the names of the animals that are related to the tiger and are not common

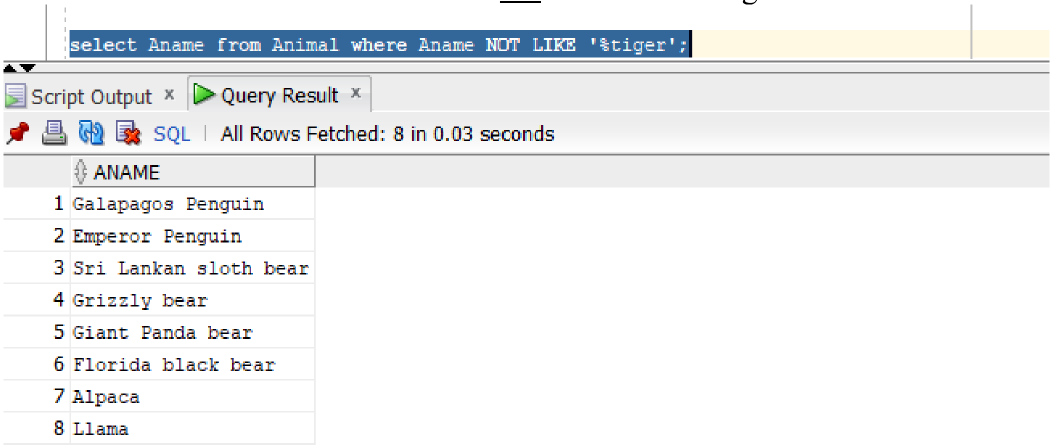


```
select Aname, ACategory from Animal where Aname LIKE '%tiger' AND NOT ACategory = 'common';
```

All Rows Fetched: 2 in 0.023 seconds

	ANAME	ACATEGORY
1	Siberian tiger	rare
2	South China tiger	exotic

4. Find the names of the animals that are <u>not</u> related to the tiger

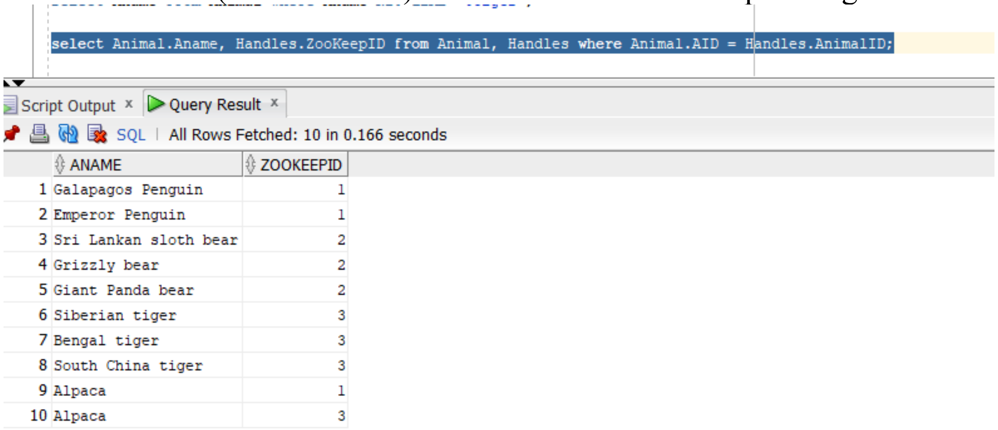


```
select Aname from Animal where Aname NOT LIKE '%tiger';
```

All Rows Fetched: 8 in 0.03 seconds

	ANAME
1	Galapagos Penguin
2	Emperor Penguin
3	Sri Lankan sloth bear
4	Grizzly bear
5	Giant Panda bear
6	Florida black bear
7	Alpaca
8	Llama

5. List the animals (animal names) and the ID of the zoo keeper assigned to them.

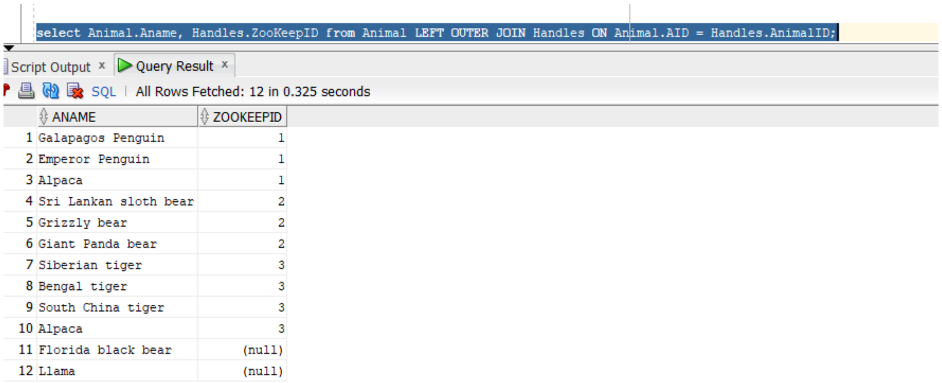


```
select Animal.Aname, Handles.ZooKeepID from Animal, Handles where Animal.AID = Handles.AnimalID;
```

All Rows Fetched: 10 in 0.166 seconds

	ANAME	ZOOKEEPID
1	Galapagos Penguin	1
2	Emperor Penguin	1
3	Sri Lankan sloth bear	2
4	Grizzly bear	2
5	Giant Panda bear	2
6	Siberian tiger	3
7	Bengal tiger	3
8	South China tiger	3
9	Alpaca	1
10	Alpaca	3

6. Now repeat the previous query and make sure that the animals without an assigned handler also appear in the answer.

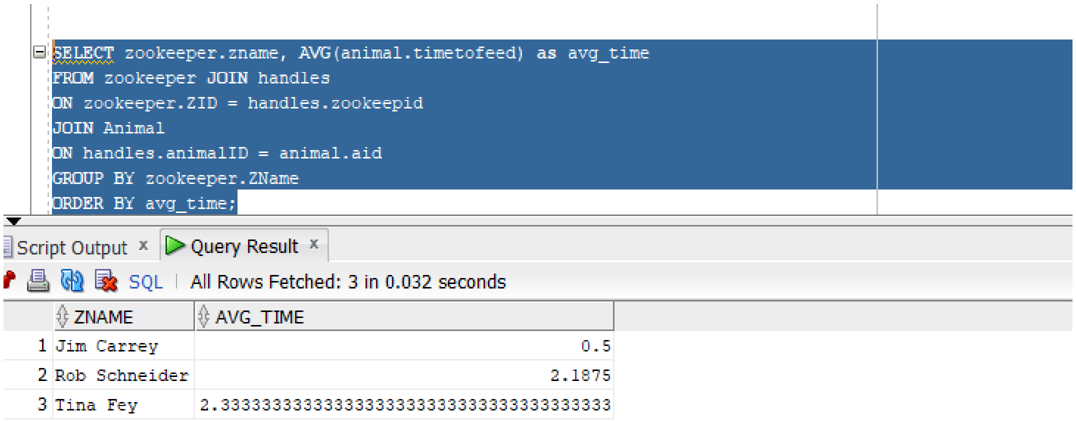


```
select Animal.Aname, Handles.ZooKeepID from Animal LEFT OUTER JOIN Handles ON Animal.AID = Handles.AnimalID;
```

All Rows Fetched: 12 in 0.325 seconds

	ANAME	ZOOKEEPID
1	Galapagos Penguin	1
2	Emperor Penguin	1
3	Alpaca	1
4	Sri Lankan sloth bear	2
5	Grizzly bear	2
6	Giant Panda bear	2
7	Siberian tiger	3
8	Bengal tiger	3
9	South China tiger	3
10	Alpaca	3
11	Florida black bear	(null)
12	Llama	(null)

7. Report, for every zoo keeper name, the average number of hours they spend feeding all animals in their care.

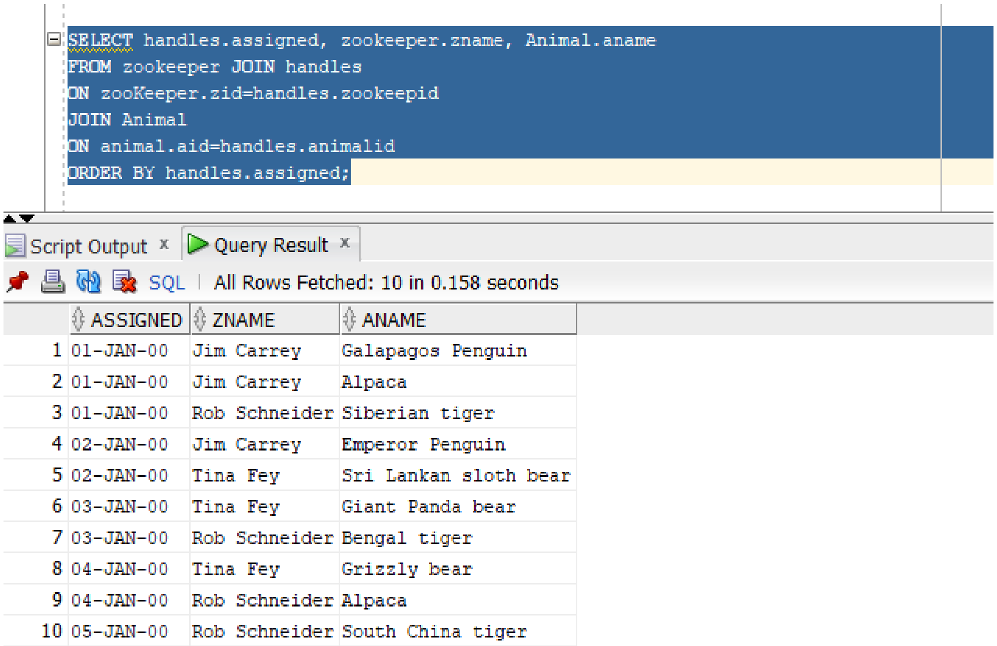


```
SELECT zookeeper.zname, AVG(animal.timetofeed) as avg_time
FROM zookeeper JOIN handles
ON zookeeper.ZID = handles.zookeepid
JOIN Animal
ON handles.animalID = animal.aid
GROUP BY zookeeper.ZName
ORDER BY avg_time;
```

All Rows Fetched: 3 in 0.032 seconds

	ZNAME	AVG_TIME
1	Jim Carrey	0.5
2	Rob Schneider	2.1875
3	Tina Fey	2.3333333333333333333333333333333333333

8. Report every handling assignment (as a list of assignment date, zoo keeper name and animal name). Sort the result of the query by the assignment date in an ascending order.



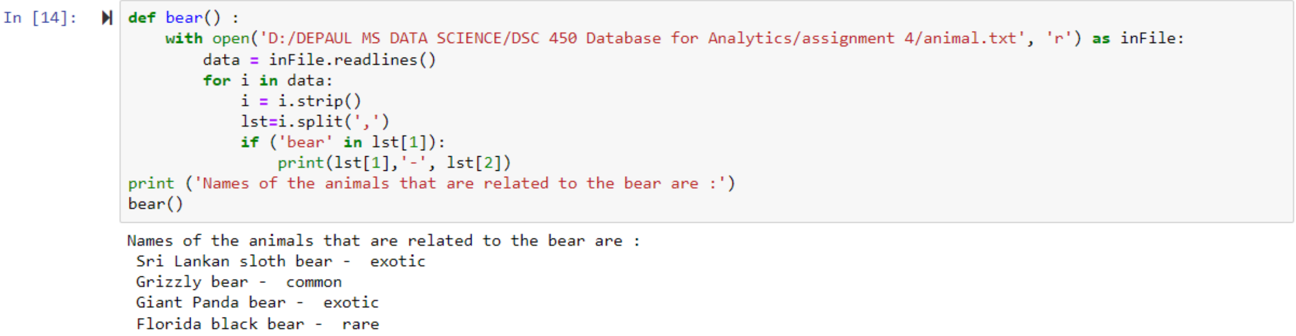
```sql
SELECT handles.assigned, zookeeper.zname, Animal.aname
FROM zookeeper JOIN handles
ON zooKeeper.zid=handles.zookeepid
JOIN Animal
ON animal.aid=handles.animalid
ORDER BY handles.assigned;
```

All Rows Fetched: 10 in 0.158 seconds

	ASSIGNED	ZNAME	ANAME
1	01-JAN-00	Jim Carrey	Galapagos Penguin
2	01-JAN-00	Jim Carrey	Alpaca
3	01-JAN-00	Rob Schneider	Siberian tiger
4	02-JAN-00	Jim Carrey	Emperor Penguin
5	02-JAN-00	Tina Fey	Sri Lankan sloth bear
6	03-JAN-00	Tina Fey	Giant Panda bear
7	03-JAN-00	Rob Schneider	Bengal tiger
8	04-JAN-00	Tina Fey	Grizzly bear
9	04-JAN-00	Rob Schneider	Alpaca
10	05-JAN-00	Rob Schneider	South China tiger

B) Repeat the following queries using python (i.e., by reading data from animal.txt, without using a database or SQL language). The idea is to replicate what a database does when the query executes.

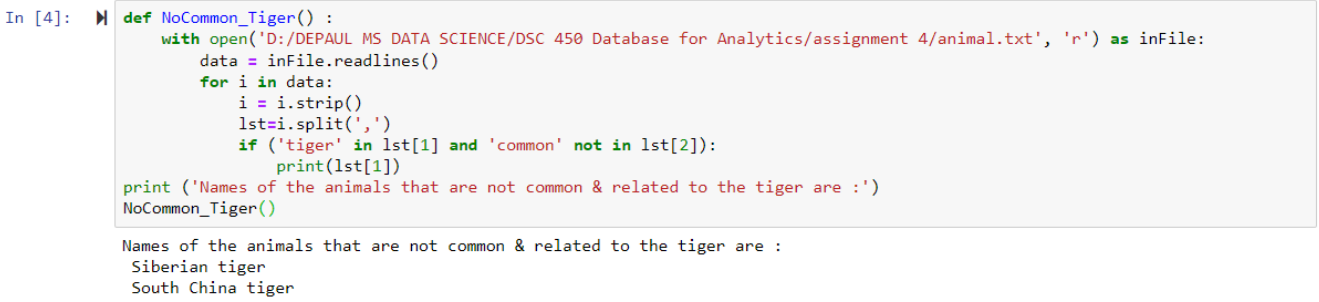
1. Find the animal names and categories for animals related to a bear



```python
def bear() :
    with open('D:/DEPAUL MS DATA SCIENCE/DSC 450 Database for Analytics/assignment 4/animal.txt', 'r') as inFile:
        data = inFile.readlines()
        for i in data:
            i = i.strip()
            lst=i.split(',')
            if ('bear' in lst[1]):
                print(lst[1],'-', lst[2])
print ('Names of the animals that are related to the bear are :')
bear()
```

```
Names of the animals that are related to the bear are :
 Sri Lankan sloth bear -  exotic
 Grizzly bear -  common
 Giant Panda bear -  exotic
 Florida black bear -  rare
```

2. Find the names of the animals that are related to the tiger and are not common



```python
def NoCommon_Tiger() :
    with open('D:/DEPAUL MS DATA SCIENCE/DSC 450 Database for Analytics/assignment 4/animal.txt', 'r') as inFile:
        data = inFile.readlines()
        for i in data:
            i = i.strip()
            lst=i.split(',')
            if ('tiger' in lst[1] and 'common' not in lst[2]):
                print(lst[1])
print ('Names of the animals that are not common & related to the tiger are :')
NoCommon_Tiger()
```

```
Names of the animals that are not common & related to the tiger are :
 Siberian tiger
 South China tiger
```

Part 2

A) You are given a following schema in 1NF:
(First, Last, Address, Job, Salary, Assistant) and the following set of functional dependencies:

First, Last → Address
Job → Salary, Assistant

Decompose the schema to make sure it is in Third Normal Form (3NF).

Employees (First,Last, Address)

Jobs(Job, Salary, Assistant)

Employees_Data(first, Last, Job)

In Employee_Data, first,last are foregin keys to Employees table &
In Employee_Data, job is foreign key to jobs table.

B) Write the necessary SQL DDL statements (CREATE TABLE) to define these the tables you created

Employee

```
CREATE TABLE Employee
(
 Firstname VARCHAR2(20),
 Lastname VARCHAR2(20),
 Address VARCHAR2(50),

 CONSTRAINT Employee_PK
 PRIMARY KEY (Firstname,Lastname)
);
```

Jobs

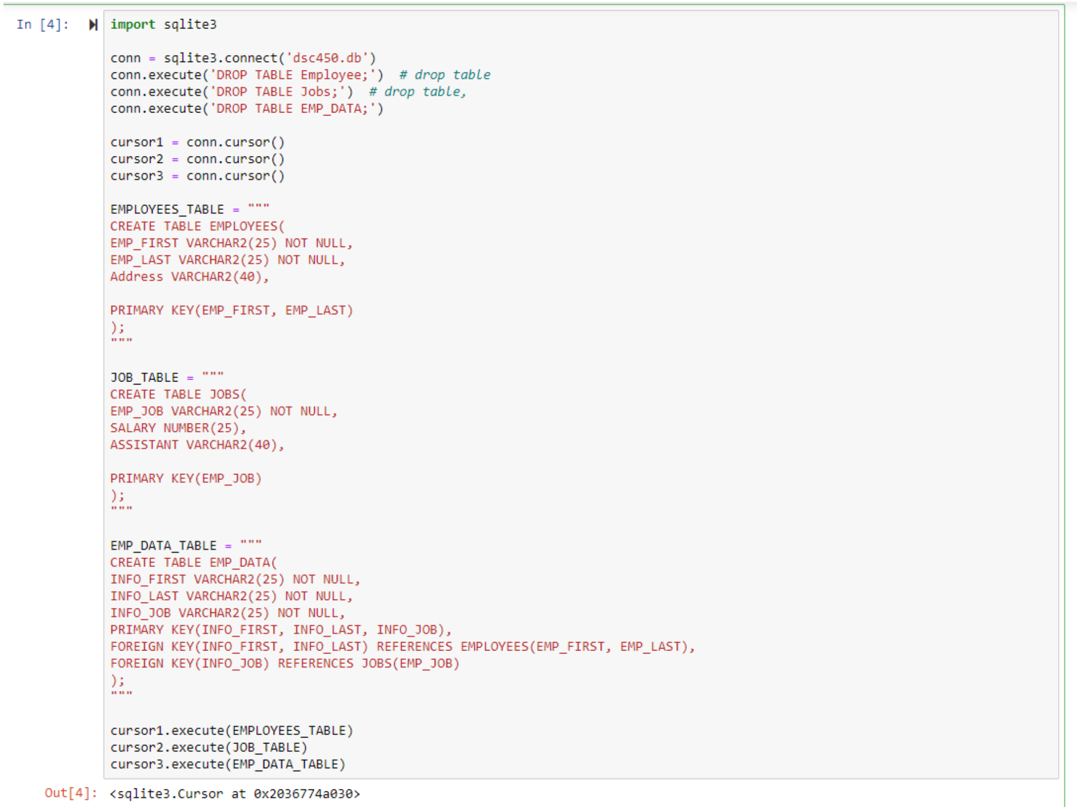
```
CREATE TABLE Jobs
(
 Job VARCHAR2(20) ,
 Salary NUMBER(9),
 Assistant VARCHAR2(15),

 CONSTRAINT Jobs_PK
 PRIMARY KEY (Job));
```

Employee_Data

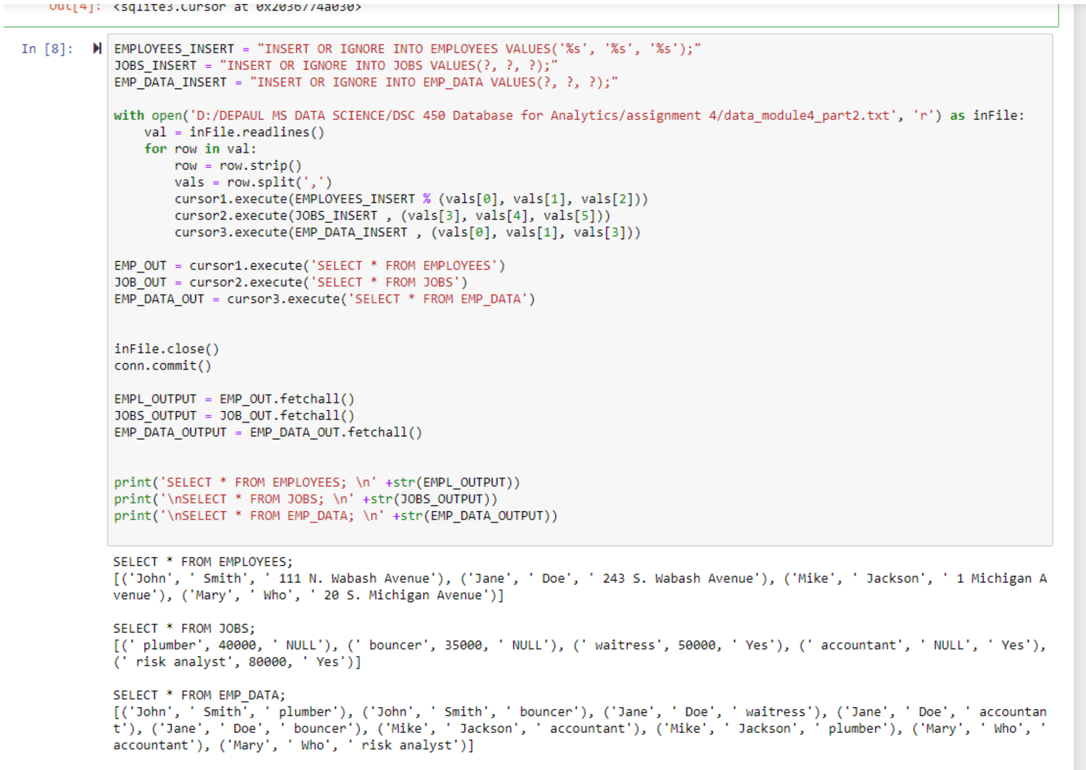
```
CREATE TABLE Employee_Data
(
 First VARCHAR2(20),
 Last VARCHAR2(20),
 Job VARCHAR2(20),

 CONSTRAINT Employee_Data_PK
 PRIMARY KEY (First, Last, Job),

 CONSTRAINT Employee_Data_FK1
 FOREIGN KEY (First, Last)
 REFERENCES Employee (Firstname,Lastname),

 CONSTRAINT Employee_Data_FK3
 FOREIGN KEY (Job)
 REFERENCES Jobs (Job)
);
```

C) Write a python script that is going to create your tables and populate them with data automatically from data_module4_part2.txt (file attached). You do not have to use executemany, your python code can load data row-by-row. Make sure that you are inserting a proper NULL into the database. HINT: You can use INSERT OR IGNORE statement (instead of a regular INSERT statement) in SQLite to skip over duplicate primary key inserts without throwing an error.

For example:
cursor.execute("INSERT OR IGNORE INTO Animal VALUES(?,?,?,?)", [11, 'Llama', None, 3.5]);
would automatically ignore the insert if animal with ID 11 already exists in the database and insert a NULL into the third column. If you use ‘NULL’ value instead, animal category would be set to the 4-character string ‘NULL’



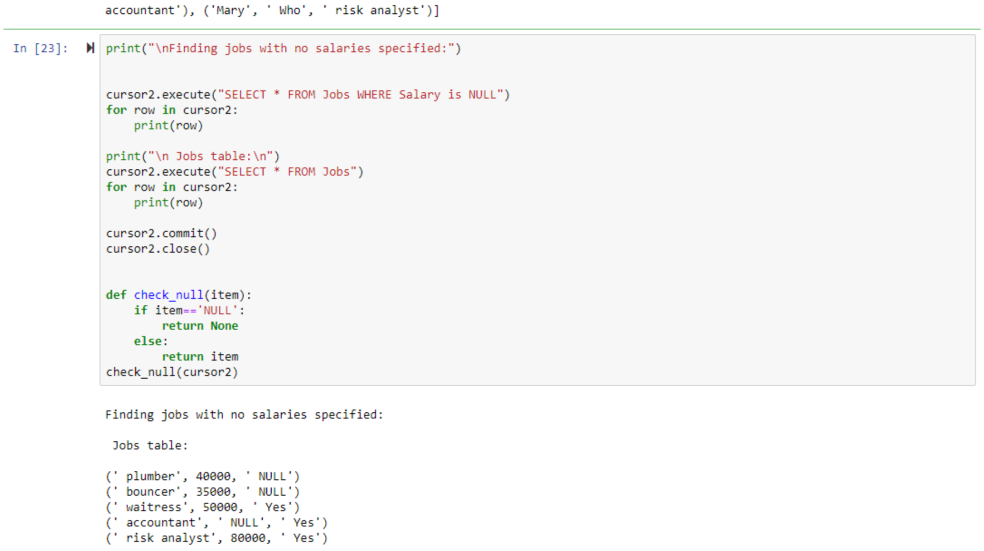
```
In [4]:  import sqlite3

         conn = sqlite3.connect('dsc450.db')
         conn.execute('DROP TABLE Employee;')  # drop table
         conn.execute('DROP TABLE Jobs;')  # drop table,
         conn.execute('DROP TABLE EMP_DATA;')

         cursor1 = conn.cursor()
         cursor2 = conn.cursor()
         cursor3 = conn.cursor()

         EMPLOYEES_TABLE = """
         CREATE TABLE EMPLOYEES(
         EMP_FIRST VARCHAR2(25) NOT NULL,
         EMP_LAST VARCHAR2(25) NOT NULL,
         Address VARCHAR2(40),

         PRIMARY KEY(EMP_FIRST, EMP_LAST)
         );
         """

         JOB_TABLE = """
         CREATE TABLE JOBS(
         EMP_JOB VARCHAR2(25) NOT NULL,
         SALARY NUMBER(25),
         ASSISTANT VARCHAR2(40),

         PRIMARY KEY(EMP_JOB)
         );
         """

         EMP_DATA_TABLE = """
         CREATE TABLE EMP_DATA(
         INFO_FIRST VARCHAR2(25) NOT NULL,
         INFO_LAST VARCHAR2(25) NOT NULL,
         INFO_JOB VARCHAR2(25) NOT NULL,
         PRIMARY KEY(INFO_FIRST, INFO_LAST, INFO_JOB),
         FOREIGN KEY(INFO_FIRST, INFO_LAST) REFERENCES EMPLOYEES(EMP_FIRST, EMP_LAST),
         FOREIGN KEY(INFO_JOB) REFERENCES JOBS(EMP_JOB)
         );
         """

         cursor1.execute(EMPLOYEES_TABLE)
         cursor2.execute(JOB_TABLE)
         cursor3.execute(EMP_DATA_TABLE)

Out[4]:  <sqlite3.Cursor at 0x2036774a030>
```



Out[4]: <sqlite3.Cursor at 0x2036774a030>

```
In [8]:  EMPLOYEES_INSERT = "INSERT OR IGNORE INTO EMPLOYEES VALUES('%s', '%s', '%s');"
         JOBS_INSERT = "INSERT OR IGNORE INTO JOBS VALUES(?, ?, ?);"
         EMP_DATA_INSERT = "INSERT OR IGNORE INTO EMP_DATA VALUES(?, ?, ?);"

         with open('D:/DEPAUL MS DATA SCIENCE/DSC 450 Database for Analytics/assignment 4/data_module4_part2.txt', 'r') as inFile:
             val = inFile.readlines()
             for row in val:
                 row = row.strip()
                 vals = row.split(',')
                 cursor1.execute(EMPLOYEES_INSERT % (vals[0], vals[1], vals[2]))
                 cursor2.execute(JOBS_INSERT , (vals[3], vals[4], vals[5]))
                 cursor3.execute(EMP_DATA_INSERT , (vals[0], vals[1], vals[3]))

         EMP_OUT = cursor1.execute('SELECT * FROM EMPLOYEES')
         JOB_OUT = cursor2.execute('SELECT * FROM JOBS')
         EMP_DATA_OUT = cursor3.execute('SELECT * FROM EMP_DATA')


         inFile.close()
         conn.commit()

         EMPL_OUTPUT = EMP_OUT.fetchall()
         JOBS_OUTPUT = JOB_OUT.fetchall()
         EMP_DATA_OUTPUT = EMP_DATA_OUT.fetchall()


         print('SELECT * FROM EMPLOYEES; \n' +str(EMPL_OUTPUT))
         print('\nSELECT * FROM JOBS; \n' +str(JOBS_OUTPUT))
         print('\nSELECT * FROM EMP_DATA; \n' +str(EMP_DATA_OUTPUT))

         SELECT * FROM EMPLOYEES;
         [('John', ' Smith', ' 111 N. Wabash Avenue'), ('Jane', ' Doe', ' 243 S. Wabash Avenue'), ('Mike', ' Jackson', ' 1 Michigan A
         venue'), ('Mary', ' Who', ' 20 S. Michigan Avenue')]

         SELECT * FROM JOBS;
         [(' plumber', 40000, ' NULL'), (' bouncer', 35000, ' NULL'), (' waitress', 50000, ' Yes'), (' accountant', ' NULL', ' Yes'),
         (' risk analyst', 80000, ' Yes')]

         SELECT * FROM EMP_DATA;
         [('John', ' Smith', ' plumber'), ('John', ' Smith', ' bouncer'), ('Jane', ' Doe', ' waitress'), ('Jane', ' Doe', ' accountan
         t'), ('Jane', ' Doe', ' bouncer'), ('Mike', ' Jackson', ' accountant'), ('Mike', ' Jackson', ' plumber'), ('Mary', ' Who', '
         accountant'), ('Mary', ' Who', ' risk analyst')]
```

D) Verify that your NULLS are loaded correctly, by finding all jobs with no salary specified using **Salary IS NULL** condition.



```
accountant'), ('Mary', ' Who', ' risk analyst')]
```

```
In [23]: print("\nFinding jobs with no salaries specified:")


         cursor2.execute("SELECT * FROM Jobs WHERE Salary is NULL")
         for row in cursor2:
             print(row)

         print("\n Jobs table:\n")
         cursor2.execute("SELECT * FROM Jobs")
         for row in cursor2:
             print(row)

         cursor2.commit()
         cursor2.close()


         def check_null(item):
             if item=='NULL':
                 return None
             else:
                 return item
         check_null(cursor2)
```

```
Finding jobs with no salaries specified:

 Jobs table:

(' plumber', 40000, ' NULL')
(' bouncer', 35000, ' NULL')
(' waitress', 50000, ' Yes')
(' accountant', ' NULL', ' Yes')
(' risk analyst', 80000, ' Yes')
```