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)

```
--1
SELECT
* FROM Animal
WHERE acategory = 'rare'
ORDER BY timetofeed;
```

	AID	ANAME	ACATEGORY	TIMETOFEED
1	2	Emperor Penguin	rare	0.75
2	6	Florida black bear	rare	1.75
3	7	Siberian tiger	rare	3.5

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

```
--2
SELECT aname, acategory
FROM Animal
WHERE aname LIKE '%bear';
```

	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

```
--3
SELECT aname
FROM Animal
WHERE aname LIKE '%tiger' AND acategory !='common';
```

	ANAME
1	Siberian tiger
2	South China tiger

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

```
--4
SELECT aname
FROM Animal
WHERE aname NOT LIKE '%tiger'
```

	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.

```
--5
SELECT aname,zookeepid
FROM Animal, Handles
WHERE Animal.aid = handles.animalid;
```

	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.

```
--6
SELECT aname,zookeepid
FROM Animal LEFT OUTER JOIN Handles
ON animal.aid = handles.animalid;
```

	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.

```
--7
SELECT zname, AVG(timetofeed)
FROM Animal,Handles,Zookeeper
WHERE handles.zookeepid = zookeeper.zid
GROUP BY zname;
```

	ZNAME	AVG(TIMETOFEED)
1	Tina Fey	2.022727272727272727272727272727272727
2	Jim Carrey	2.022727272727272727272727272727272727
3	Rob Schneider	2.022727272727272727272727272727272727

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.

```
--8
SELECT assigned, zname, aname
FROM Animal,Handles,Zookeeper
WHERE handles.zookeepid = zookeeper.zid
ORDER BY assigned;
```

	ASSIGNED	ZNAME	ANAME
1	01-JAN-00	Jim Carrey	Galapagos Penguin
2	01-JAN-00	Rob Schneider	Llama
3	01-JAN-00	Jim Carrey	Sri Lankan sloth bear
4	01-JAN-00	Jim Carrey	Grizzly bear
5	01-JAN-00	Jim Carrey	Giant Panda bear
6	01-JAN-00	Jim Carrey	Florida black bear
7	01-JAN-00	Jim Carrey	Siberian tiger
8	01-JAN-00	Jim Carrey	Bengal tiger
9	01-JAN-00	Jim Carrey	South China tiger
10	01-JAN-00	Jim Carrey	Alpaca
11	01-JAN-00	Jim Carrey	Llama
12	01-JAN-00	Jim Carrey	Emperor Penguin
13	01-JAN-00	Rob Schneider	Alpaca
14	01-JAN-00	Rob Schneider	South China tiger
15	01-JAN-00	Rob Schneider	Bengal tiger
16	01-JAN-00	Rob Schneider	Siberian tiger
17	01-JAN-00	Rob Schneider	Florida black bear

	ASSIGNED	ZNAME	ANAME
16	01-JAN-00	Rob Schneider	Siberian tiger
17	01-JAN-00	Rob Schneider	Florida black bear
18	01-JAN-00	Rob Schneider	Giant Panda bear
19	01-JAN-00	Rob Schneider	Grizzly bear
20	01-JAN-00	Rob Schneider	Sri Lankan sloth bear
21	01-JAN-00	Rob Schneider	Emperor Penguin
22	01-JAN-00	Rob Schneider	Galapagos Penguin
23	01-JAN-00	Jim Carrey	Llama
24	01-JAN-00	Jim Carrey	Alpaca
25	01-JAN-00	Jim Carrey	South China tiger
26	01-JAN-00	Jim Carrey	Bengal tiger
27	01-JAN-00	Jim Carrey	Siberian tiger
28	01-JAN-00	Jim Carrey	Florida black bear
29	01-JAN-00	Jim Carrey	Giant Panda bear
30	01-JAN-00	Jim Carrey	Grizzly bear
31	01-JAN-00	Jim Carrey	Sri Lankan sloth bear
32	01-JAN-00	Jim Carrey	Emperor Penguin

Goes on for 110 rows^

B) Repeat the following queries using python (i.e., by reading data from animal.txt, without using a database)

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

```
fd = open("animal.txt",'r')
animal = fd.readlines()
#1
for row in animal:
    r1=row.strip()
    if 'tiger' in r1.split(',')[1] and 'common' not in r1.split(',')[2]:
        print(r1.split(',')[1])
```

```
>>> #1
... for row in animal:
...     r1=row.strip()
...     if 'tiger' in r1.split(',')[1] and 'common' not in r1.split(',')[2]:
...         print(r1.split(',')[1])
...
Siberian tiger
South China tiger
```

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

```
#2
for row in animal:
    r1=row.strip()
    if 'tiger' not in r1.split(',')[1]:
        print(r1.split(',')[1])
```

```
>>> #2
... for row in animal:
...     r1=row.strip()
...     if 'tiger' not in r1.split(',')[1]:
...         print(r1.split(',')[1])
...
 Galapagos Penguin
 Emperor Penguin
 Sri Lankan sloth bear
 Grizzly bear
 Giant Panda bear
 Florida black bear
 Alpaca
 Llama
```

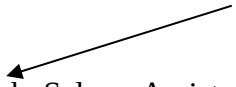
Part 2

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).

Job (First, Last, Address, *Job*)



Position (Job, Salary, Assistant)

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

```
drop table position;
drop table employee;

Create Table Position (
    Job VARCHAR2(30),
    Salary NUMBER (10,2),
    Assistant VARCHAR2(40),

    Constraint Position_PK
        PRIMARY KEY (Job)
);

CREATE TABLE Employee (
    FirstName VARCHAR2(40),
    LastName  VARCHAR2(40),
    Address   VARCHAR2(60),
    Job       VARCHAR2(30),

    Constraint Employee_PK
        PRIMARY KEY(FirstName, LastName),

    Constraint Employee_FK
        FOREIGN KEY (Job)
            REFERENCES Position(Job)
);
```


```
INSERT INTO Position VALUES('Lead Tenor Sax',150000.05, 'Miles');
INSERT INTO Employee VALUES('John','Coltrane','123 Blue Train Rd. Detroit, MI 48127','Lead Tenor Sax');

SELECT * FROM Position;
SELECT * FROM Employee;
```

	JOB	SALARY	ASSISTANT
1	Lead Tenor Sax	150000.05	YES

FIRSTNAME	LASTNAME	ADDRESS	JOB
John	Coltrane	123 Blue Train Rd. Detroit, MI 48127	Lead Tenor Sax

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’

```
file = fd.read()
allDataLine = fd.readlines()
lstP=[]
lstE=[]
for row in allDataLine:
    r1 = row.strip()
    Pos = r1.split(',')[3:6]
    Emp = r1.split(',')[0:4]
    #print(Emp)
    lstP.append(Pos)
    lstE.append(Emp)
lstP
lstE


import sqlite3
conn = sqlite3.connect('dsc450.db')
cursor = conn.cursor()

cursor.execute(Position) #position tbl
cursor.execute(Employee) #employee tbl

P = cursor.executemany("INSERT OR IGNORE INTO Position VALUES (?,?,?)",lstP)
E = cursor.executemany("INSERT OR IGNORE INTO Employee VALUES (?,?,?,?)",lstE)

#query for fun
query1 = '''SELECT * FROM Employee'''

for row in cursor.execute(query1).fetchall():
    print(row)
```

```
Position = '''
Create Table Position (
    Job VARCHAR2(30),
    Salary NUMBER,
    Assistant VARCHAR2(5),

    Constraint Position_PK
        PRIMARY KEY (Job)
);
'''

Employee = '''
CREATE TABLE Employee (
    FirstName VARCHAR2(40),
    LastName  VARCHAR2(40),
    Address   VARCHAR2(60),
    Job       VARCHAR2(30),

    Constraint Employee_PK
        PRIMARY KEY(FirstName, LastName),

    Constraint Employee_FK
        FOREIGN KEY (Job)
            REFERENCES Position(Job)
);
'''
```

```
...
>>> query1 = '''SELECT * FROM Employee'''
...
>>> query2 = '''SELECT * FROM Position'''
...
>>> for row in cursor.execute(query1).fetchall():
...     print(row)
...
('John', ' Smith', ' 111 N. Wabash Avenue', ' plumber')
('Jane', ' Doe', ' 243 S. Wabash Avenue', ' waitress')
('Mike', ' Jackson', ' 1 Michigan Avenue', ' accountant')
('Mary', ' Who', ' 20 S. Michigan Avenue', ' accountant')
>>> for row in cursor.execute(query2).fetchall():
...     print(row)
...
(' plumber', 40000, ' NULL')
(' bouncer', 35000, ' NULL')
(' waitress', 50000, ' Yes')
(' accountant', 42000, ' Yes')
(' risk analyst', 80000, ' Yes')
```